Diss. ETH No. 16106

Time Synchronization and Localization in Sensor Networks

A dissertation submitted to the
Swiss Federal Institute of Technology Zurich (ETH Zurich)

for the degree of
Dr. sc. ETH Zurich

presented by
Kay Römer
Diplom-Informatiker, University of Frankfurt/Main, Germany
born June 16, 1972
citizen of Germany

accepted on the recommendation of
Prof. Dr. Friedemann Mattern, examiner
Prof. Dr. Paul J. M. Havinga, co-examiner

2005

Abstract

So-called sensor nodes combine means for sensing environmental parameters, processors, wireless communication capabilities, and autonomous power supply in a single compact device. Networks of these untethered devices can be deployed unobtrusively in the physical environment in order to monitor a wide variety of real-world phenomena with unprecedented quality and scale while only marginally disturbing the observed physical processes.

Due to the close integration of sensor networks with the real world, the categories time and location are fundamental for many applications of sensor networks, for example to interpret sensing results (e.g., *where* and *when* did an event occur) or for coordination among sensor nodes (e.g., *which* nodes can *when* be switched to idle mode). Hence, *time synchronization* and *sensor node localization* are fundamental and closely related services in sensor networks.

Existing solutions for these two basic services have been based on a rather narrow notion of a sensor network as a large-scale, ad hoc, multi-hop, unpartitioned network of largely homogeneous, tiny, resource-constrained, mostly immobile sensor nodes that would be randomly deployed in the area of interest. However, recently developed prototypical applications indicate that this narrow definition does not cover a significant portion of the application domain of wireless sensor networks.

Our thesis is that *applications of sensor networks span a whole design space with many important dimensions. Existing solutions for time synchronization and node localization do not cover important parts of this design space. Substantially different approaches are required to support these regions adequately. Such solutions can actually be provided.*

We support this thesis by proposing a design space of wireless sensor networks where concrete applications can be located at different points of the space. We identify two important regions in the design space that are not appropriately supported by existing methods for time synchronization and node localization. We also propose, implement, and evaluate new solutions that cover these regions. The practical feasibility of our approaches is demonstrated by means of a typical sensor network application which requires time synchronization and node localization.

Our approach to time synchronization supports applications where network connectivity is intermittent. The idea underlying our *Time-Stamp Synchronization* method is to avoid proactive synchronization of the clocks of all nodes in a network. Instead, the clocks of the sensor nodes run unsynchronized, each defining its own local time scale. Only if clock readings are exchanged among nodes as time stamps

contained in network messages, these time stamps are transformed from the time scale of the sender to the time scale of the receiver. This approach is scalable, since time is only synchronized on demand where and when needed by the application. The approach is also resource efficient, since it piggybacks on existing message exchanges.

Our approach to node localization supports tiny sensor nodes known as Smart Dust. The *Lighthouse Location System* is based on a single beacon device that emits particular optical signal patterns. Sensor nodes can autonomously infer their location by passively observing these signals. This approach is scalable, since each node infers its location independent of other nodes. A single beacon device emits long-range signals in broadcast mode and can support arbitrary network densities. The approach is resource efficient, since the sensor nodes do not actively emit any signals. Only a tiny, energy-efficient optical receiver is needed to infer locations.

Zusammenfassung

Sensoren zur Erfassung von Umweltparametern, Prozessoren, drahtlose Kommunikationseinheiten sowie autarke Energiequellen sind in sogenannten Sensorknoten auf kleinstem Raum integriert. Netze aus vielen solchen Knoten können unaufdringlich in die Alltagswelt ausgebracht werden, um eine Reihe verschiedener Umweltphänomene grossräuming und mit hoher Genauigkeit zu erfassen, ohne die beobachteten Vorgänge wesentlich zu beeinflussen.

Aufgrund der Einbettung von Sensornetzen in die reale Welt spielen die Kategorien Raum und Zeit eine fundamentale Rolle für viele Anwendungen, beispielsweise zur Interpretation von Beobachtungen (z.B. *wo* und *wann* wurde ein Ereignis festgestellt) oder für die Koordination der Sensorknoten untereinander (z.B. *welche* Knoten können *wann* in einen energiesparenden Schlafzustand geschaltet werden). Daher sind *Zeitsynchronisation* und *Lokalisierung* von Sensorknoten grundlegende und eng verwandte Dienste in Sensornetzen.

Bestehende Ansätze zur Realisierung dieser Dienste gehen von einer vergleichsweise engen Definition eines Sensornetzes aus, derzufolge ein Sensornetz aus einer sehr grossen Zahl von homogenen, winzigen und daher ressourcenbeschränkten Knoten besteht, die vorwiegend immobil sind, nachdem sie zufällig im Zielgebiet verteilt wurden. Ferner geht man davon aus, das Sensornetze unpartitionierte Multi-Hop-Ad-Hoc-Netze sind. In jüngerer Zeit wurde jedoch eine Vielzahl prototypischer Anwendungen von Sensornetzen vorgestellt, denen eine solche enge Definition nicht gerecht wird.

Unsere These ist daher, dass *Applikationen von Sensornetzen einen umfangreichen Entwurfsraum aufspannen, der eine Vielzahl wichtiger Dimensionen umfasst. Bisher existierende Methoden zur Zeitsynchronisation und Lokalisierung decken wichtige Bereiche dieses Entwurfsraums nicht ab. Vielmehr benötigt man neuartige Herangehensweisen, um diese Bereiche adäquat zu unterstützen. Entsprechende Techniken können tatsächlich bereitgestellt werden.*

Wir untermauern diese These, indem wir den Entwurfsraum von Sensornetzen explizit machen und zeigen, dass konkrete Applikationen tatsächlich verschiedenen Punkten in diesem Raum zugeordnet werden können. Wir identifizieren zwei spezifische Bereiche im Entwurfsraum, welche nicht hinreichend durch bestehende Ansätze zur Zeitsynchronisation und Lokalisierung unterstützt werden. Um diese Bereiche abzudecken, schlagen wir neue Lösungsansätze vor, zeigen prototypische Realisierungen auf und evaluieren diese. Die praktische Umsetzbarkeit dieser Methoden zeigen wir anhand einer konkreten Applikation, die Synchronisation und

Lokalisierung voraussetzt.

Unser Ansatz zur Zeitsynchronisation unterstützt Anwendungsszenarien, in denen Netzverbindungen nur sporadisch bestehen. Die grundlegende Idee für das Verfahren der *Zeitstempelsynchronisation* besteht darin, die Uhren der Sensorknoten *nicht* zu synchronisieren, so dass die lokale Uhr eines jeden Knotens eine unabhängige Zeitskala definiert. Zeitstempel, die durch Auslesen der lokalen Uhr entstehen, haben daher zunächst nur lokale Gültigkeit. Wird ein solcher Zeitstempel jedoch als Teil einer Nachricht im Netz verschickt, so wird dabei der Zeitstempel von der Zeitskala des Senders in die Zeitskala des Empfängers transformiert. Dieser Ansatz ist skalierbar, da Synchronisation nur dann stattfindet, wenn sie tatsächlich von der Applikation benötigt wird. Ferner kann diese Methode effizient implementiert werden, da die für die Zeitstempeltransformation notwendige Kommunikation in vielen Fällen Huckepack auf bereits existierenden Nachrichten realisiert werden kann.

Unser Ansatz zur Lokalisierung unterstützt winzige, sehr ressourcenarme Sensorknoten, die unter dem Namen "Smart Dust" bekannt sind. Unser Verfahren mit dem Namen *Leuchtturmlokalisierung* verwendet eine spezielle Basisstation, die spezifische optische Signale aussendet. Sensorknoten können allein durch passive Beobachtung dieser Signale autonom ihre Position mit hoher Genauigkeit bestimmen. Dieser Ansatz ist skalierbar, da jeder Knoten völlig unabhängig von anderen Knoten seine Position bestimmt. Eine einzige Basistation kann daher beliebig dichten Netzen zur Lokalisierung dienen. Da die Sensorknoten für die Lokalisierung selbst keinerlei Signale aussenden müssen, ist das Verfahren auf der Seite der Sensorknoten sehr ressourceneffizient. Sensorknoten benötigen nur einen einfachen optischen Empfänger, der auf kleinstem Raum realisiert werden kann.

Contents

Chapter 1

Introduction

Enabled by technological advancements in wireless communications and embedded computing, wireless sensor networks were first considered for military applications in the 1980-ies, where large-scale wireless networks of autonomous sensor nodes would enable the unobtrusive observation of events in the real-world. Since then, the use of sensor networks has also been considered for various civil application domains. This thesis is devoted to two fundamental services required by sensor networks: time synchronization and node localization.

In this inaugural chapter, we introduce the research area of wireless sensor networks, motivate the need for time synchronization and localization in sensor networks, and give a brief overview of the main contributions of our work. We conclude this chapter with an overview of the remainder of this thesis.

1.1 Motivation

In the late 1980-ies, technology advanced to a stage where it became possible to build relatively small, battery-powered computing devices equipped with sensors and wireless communication with *manageable effort and cost* by leveraging *off-the-shelf hardware components*. While systems with similar functionality had been built earlier, these required costly custom hardware design processes or exhibited a power consumption that did not allow battery operation for longer periods of time.

Having observed the speed of technological advancements over the past, one could envision at that time that in the near future it would be possible to build even smaller untethered computing, communicating, and sensing devices with marginal cost per device. While the low per-device cost would allow mass production, small size and untetheredness would enable an unobtrusive deployment.

This prospect triggered researchers to think of implications and applications of this emerging new technology. Perhaps one of the first individuals to articulate this trend, to envision possible applications, and to speculate about societal impacts was Mark Weiser, who coined the term *Ubiquitous Computing* in his seminal article [106]. From then on, this vision was further refined and substantiated by a number of visionaries and research projects. This development was evidenced by several

new terms such as *Pervasive Computing*, *Ambient Intelligence*, and also *Wireless Sensor Networks* (WSN).

Common to these slightly different terms and underlying visions is the goal of bridging the long-standing gap between the *physical world* where we live in and the traditional *virtual world* of computers and other information-technology artifacts. The key to realization of these visions is the use of large collections of these unobtrusive networked computers that could perceive and control aspects of the real world via sensors and actuators on the one hand, and that would provide an intuitive interface to human users on the other hand. While projects classified as Ubiquitous Computing, Pervasive Computing, and Ambient Intelligence are somewhat focused on issues related to interfacing these unobtrusive networked computing devices to human users, this component is of lesser significance in projects that examine sensor networks. Rather, research on wireless sensor networks focuses on the technical aspects of observing the real world with best possible quality, using as few as possible resources, and minimizing the impact of the observation tool on the observed physical processes. We examine the subtle differences of the above research areas in more detail in Chapter 2. Our work, however, is focused on wireless sensor networks.

WSN have been initially considered for military applications, where real-world events (e.g., vehicles and troops passing) must be unobtrusively observed in inaccessible or hostile environments. For example, DARPA initiated the Distributed Sensor Networks program in the 1980-ies. For these military tasks, large numbers of sensor nodes would be deployed in the area of interest and form a wireless network to observe events in the physical environment. These long-lived, unattended networks would be unobtrusive due to the small size and untetheredness of individual nodes, could operate without the use of additional hardware infrastructure, and would be robust due to the redundant deployment of nodes. Later on, it was suggested that these features would render sensor networks a useful tool also in a number of civil application domains [35], for example as a scientific tool for environmental monitoring or in building automation. In Chapter 2 we examine a number of concrete civil applications of WSN.

Time and space are fundamental categories in the physical world. Since wireless sensor networks are a tool for observing, influencing, and reasoning about phenomena of the physical word, time and space are also of utmost importance in WSN. They are essential elements for obtaining and interpreting real-world observations (e.g., *where* and *when* did an event occur, how *large* and *fast* was an observed object), for tasking a sensor network (e.g., *where* and *when* to look for events), for interfacing wireless sensor networks with the real-world (e.g., what node *density* and sampling *frequency* is needed to observe a certain object), and for coordination among sensor nodes (e.g., *which* nodes can *when* be switched to idle mode).

There are two basic services to enable these functions: *time synchronization* and *localization of sensor nodes*. Time synchronization allows a sensor node to estimate current time with respect to a common time scale. Localization allows a node to estimate its current location with respect to a common coordinate system.

1.2 Contributions

This thesis is devoted to time synchronization and node localization in the context of wireless sensor networks. What makes the provision of these services challenging are the specific technical characteristics and requirements of wireless sensor networks and their applications. A number of earlier research projects and commonly used hardware prototypes of sensor nodes led to a rather narrow view on these characteristics and requirements, which resulted in a de facto definition of a wireless sensor network that is adopted by most research projects. Consequently, existing work on time synchronization and node localization is mostly based on this narrow view.

One of the contributions of this thesis is to show that such a narrow view on the characteristics and requirements of WSN does not meet the diversity of concrete applications of wireless sensor networks. Motivated by a study of concrete applications of WSN, we propose to replace this narrow definition with a multi-dimensional *design space* that captures influential and significant dimensions of wireless sensor networks and their applications. We substantiate the relevance of such a design space by showing that concrete prototypical applications of wireless sensor networks can indeed be located at a diverse set of points in the design space.

We show that existing approaches to time synchronization and node localization fail to cover important parts of this design space. In particular, we identify two regions in the design space which are not sufficiently supported by existing solutions. The main contribution of this thesis is to propose novel approaches to node localization and time synchronization to support these regions. We present and evaluate prototypical implementations of our solutions. In addition, we support the practical feasibility of our algorithms by incorporating them into a concrete application for tracking mobile objects with a wireless sensor network.

One further contribution of this thesis is the provision of a unified view on time synchronization and node localization in the context of wireless sensor networks. While research in these two domains has been largely separated in the past, we show that models, requirements, techniques, and algorithms of the two domains are rather similar and in some respects closely related. In particular, we point out a number of structural elements that are shared by many existing distributed algorithms for time synchronization and node localization.

The major contributions of this thesis have also been published in scientific conferences, journals, and books, most notably in [34, 80, 82, 83, 84, 85, 86].

1.3 Structure

This thesis first discusses general aspects of wireless sensor networks, before presenting a unified framework for the discussion of time and space. Based on this framework, we discuss time synchronization and node localization separately. We then rejoin our discussion on time and space by showing how our solutions are

employed in a common prototypical application. In more detail, the thesis is structured as follows:

Chapter 2 is devoted to general aspects of wireless sensor networks. We characterize wireless sensor networks by showing how they draw from other research domains and point out important differences with respect to these domains. We then propose the design space of wireless sensor networks and justify it by showing that existing applications do indeed cover different regions in this space. We discuss different classes of sensor-node prototypes and show how they cover various regions of the design space. We then discuss how different regions in the design space are associated with different technical challenges and conclude the chapter with design principles that are helpful in dealing with these challenges.

Chapter 3 presents a unified framework for the discussion of aspects related to space and time in sensor networks. We first present applications of space and time in sensor networks, before developing a common model for time synchronization and node localization. Based on this common model, we present requirements and possible conceptual approaches for time synchronization and node localization. Then we examine the structure of distributed algorithms for time synchronization and node localization, pointing out five important structural elements that can be found in most distributed algorithms of both domains. Finally, we discuss various limitations and trade-offs of these distributed algorithms with respect to the technical challenges presented in Chapter 2.

Chapters 4 and 5 are devoted to in-detail examinations of time synchronization and node localization, respectively. The structure of these two chapters is very similar. Following the framework developed in Chapter 3, we first review fundamental concepts and techniques. Based on these concepts, we present and discuss concrete existing algorithms. We then show how these approaches fail to meet the requirements of certain important regions in the design space that was developed in Chapter 2. Finally, we present and evaluate our solutions for these specific regions in the design space.

In Chapter 6 we show the practical feasibility of our solutions for time synchronization and node localization by means of a concrete prototypical application.

Chapter 7 concludes this thesis by summarizing the results, by discussing limitations, and by providing an outlook on future work.

Chapter 2

Wireless Sensor Networks

Research on wireless sensor networks goes back to a number of US-based research projects, where the use of large networks of tiny wireless sensor devices was explored in a military domain. Initial work mainly focused on the development of hardware prototypes and energy-efficient networking protocols. These early efforts established a de facto definition of a wireless sensor network as a large-scale, wireless, ad hoc, multi-hop, unpartitioned network of homogeneous, tiny, mostly immobile sensor nodes that would be randomly deployed in the area of interest.

Since then, the use of wireless sensor networks has also been considered for a number of civil applications. Wireless sensor networks have been suggested as a scientific tool for better understanding real-world phenomena, as an enabling technology for making our daily life more comfortable, as a tool for improving the efficiency of industrial processes, and as a mechanism for dealing with issues such as environmental protection and law enforcement. In these application domains, wireless sensor networks are deemed a promising technology with the potential for changing the way we live by bridging the gap between the real world and the virtual world of existing information technology.

The diverse set of potential applications has two important implications. Firstly, wireless sensor networks cannot any longer be characterized by a single, narrow definition. Rather, wireless sensor networks span a broad design space with vastly varying requirements and characteristics. Secondly, wireless sensor networks have become a truly multidisciplinary domain, where close cooperation between users, application domain experts, hardware designers, and software developers is needed to realize efficient systems for specific applications.

In this chapter, we characterize wireless sensor networks in a number of different ways. We first informally define wireless sensor networks in Section 2.1, also pointing out the research areas that are influential. In Section 2.2 we make the design space of wireless sensor networks explicit by identifying its important dimensions. We consider existing and envisioned applications of wireless sensor networks in Section 2.4 and show that these applications do indeed cover various regions in the design space. In Section 2.5 we discuss four classes of sensor node prototypes and show how these can cover different areas in the design space. Sec-

tion 2.6 is devoted to prominent technical challenges that can be found at different points in the design space. Design principles that can help mitigate these technical challenges are presented in Section 2.7.

2.1 Characterization

Sensor networks consist of sensor nodes – untethered computing devices that include a power source, a transceiver for wireless communication, a processor, memory, sensors, and potentially also actuators. Although the exact properties and capabilities of these components may vary, a common property of sensor nodes is their resource scarcity.

Multiple sensor nodes form a wireless network, whose topology and other properties do also depend on the application context. A large class of sensor networks can be characterized as multi-hop ad hoc networks, where sensor nodes do not only act as data sources, but also as routers that forward messages on behalf of other nodes, such that no additional communication infrastructure (e.g., base stations) is required for operating the network.

The sensor nodes participating in a network can vary in their capabilities and configuration. For example, sensor nodes may be equipped with different types of sensors; some sensor nodes might be equipped with a more powerful processor and more memory to perform sophisticated computations; some nodes might be connected to a other networks and can act as gateways to a background infrastructure.

Wireless sensor networks are deployed in the physical environment in order to monitor a wide variety of real-world phenomena with unprecedented quality and scale (by placing many sensor nodes close to the phenomenon of interest), while only marginally disturbing the observed physical processes (due to the unobtrusiveness of individual sensor nodes).

Using attached sensors, nodes can observe a partial state of the real world in their close physical neighborhood. By integrating observations of many sensor nodes, a more detailed and geographically extensive observation of a partial state of the real world can be obtained. Due to the relatively small effective range of sensor nodes, sensor networks often consist of many, densely deployed sensor nodes.

While individual sensor nodes have only limited functionality, the global behavior of a sensor network can be quite complex. The true value of the network is in this emergent behavior: the functionality of the whole is greater than the sum of its parts. For example, a sensor network may estimate the velocity of a moving object even though sensor nodes are not equipped with velocity sensors. Instead, velocity estimates can be obtained by correlating object sightings from spatially dispersed sensor nodes, which requires only sensors for detecting the proximity of objects.

The output of the sensor network may be used for various purposes. In the most basic form, the output is delivered to a human user for further evaluation. However, the output may also be used to control the operation of the sensor network without human intervention by enabling/disabling sensors, or by controlling

operation parameters of sensors (e.g., sampling rate, sensitivity, orientation, position). In addition, actuators may be triggered based on the output of the sensor network. Using the output of the sensor network to control sensors or actuators can effectively create a closed-loop system that strives to achieve a particular nominal condition in the sensor network or in the real world.

Sensor networks are a multidisciplinary area of research which is related to and draws from a number of other research domains. However, due to a number of novel characteristics and requirements of sensor networks, results from other research domains typically cannot be directly applied to sensor networks. Below we further characterize wireless sensor networks by discussing closely related and influential research domains. In particular, we point out how sensor networks differ from typical assumptions and models in these domains.

2.1.1 Distributed Systems

According to [7], a distributed system is an *information-processing* system that contains a number of independent *computers* that *cooperate* with one another over a *communications networks* in order to achieve a specific objective. Based on this definition, wireless sensor networks are clearly distributed systems: sensor nodes nodes cooperate by means of wireless communication in order to process information about the real world.

In the sensor network context, *computers* and *communications network* differ significantly from many traditional distributed systems. The unobtrusive deployment of large-scale sensor networks requires that sensor nodes be untethered, small, and cheap. Size and cost constraints in turn imply that sensor nodes are limited in their resources (computing, storage, communication) and energy budget. *Communication* is wireless, typically short range, low bandwidth, and unreliable. Interaction with a harsh physical environment may lead to a high degree of network dynamics (e.g., topology changes, network partitions, node failures, communication failures) typically not found in traditional distributed systems.

Traditional distributed systems are mostly decoupled from the real world. In contrast, sensor networks are inherently and closely integrated with the real world, with data about the physical environment being captured and processed automatically, online, and in real time. The input of a sensor network can be characterized as a continuous stream of data with low information density (raw sensory data contains few information per bit), with high data volume (since many nodes frequently sample their sensors), with many redundancies and correlations (since many physical phenomena are continuous in time and space), and with a high level of noise (since data is obtained by measurements using low-cost sensors). The desired output typically has opposite characteristics: high information density, low date volume, low redundancy, and high accuracy.

Sensor nodes *cooperate* with the goal of distilling the essence of information contained in a large amount of dispersed sensor readings, taking into account the limitations of individual nodes and the limitations of the communications network.

2.1.2 Ubiquitous Computing

The term "Ubiquitous Computing" [106] refers to a world where computers are unobtrusively integrated into our natural environment in order to support us intuitively in fulfilling our tasks. This vision is based on the observation that using today's general purpose computers often requires complex and non-intuitive interaction patterns. Humans have to pay special attention to the computer, are distracted from the task to be solved – ultimately making the computer a very visible component of our current world. In contrast, using a "ubiquitous computer" should not require abilities that are not immediately related to the task at hand. In addition to this smooth integration into our natural environment, ubiquitous computers are also physically embedded into our environment and interact with the physical world by means of sensors and actuators.

One possible way to approach this vision is to integrate computing and information-processing capabilities into familiar artifacts or into our physical environment – resulting in so-called "smart things" and "smart environments". Here, the term "smart" refers to an augmented functionality that is both intuitive and useful. In contrast to traditional general purpose computers, smart things and environments are highly specialized for a particular application. The computers embedded into smart things and smart environments communicate with each other or with a background infrastructure via a wireless network in order to provide complex services.

Important for the intuitive or smart behavior of ubiquitous computing systems is the notion of *context*. According to [29], context is any information that characterizes the situation of an entity (e.g., person, place, object). While this is a very broad characterization, context information (e.g., location, time, light, temperature, presence/absence of other entities) can often be circumscribed as the state of an entity and its close physical environment. In most cases, sensors are used to acquire context information. Context is a key to intuitive usage, since it is a primary source of information for adapting system behavior to the current situation and expectations of users. In the ubicomp literature, such context-driven adaptation is referred to as "context awareness".

From the above description, wireless networks of embedded sensing and computing devices are a key enabling technology for ubicomp and have been used in many prototypical ubicomp applications to date. Despite this, it is interesting to note that the ubicomp and WSN research communities are largely separated both in terms of research groups and venues for presenting research results.

One reason for this might be the somewhat different research foci of the two communities. Typical topics in ubicomp research are user-interface issues, applications and their implications, high-level aspects of context acquisition and context awareness. The WSN community often focuses on networking aspects, distributed algorithms, and distributed signal processing. One further difference is related to the use of networked sensors in ubicomp and WSN. Sensors for context acquisition in ubicomp are often incorporated into specific artifacts, resulting in a predefined and fixed association of sensors to a particular artifact. Detecting the context of

such instrumented entities typically involves only a single or few networked sensors. Deriving context information from raw sensor readings is often performed outside of the "sensor network".

In contrast, typical wireless sensor networks are embedded into the environment to monitor phenomena occurring within this environment. Often, these phenomena cannot be directly instrumented with sensors for observation. As a consequence, there may be no predefined or fixed association of sensor nodes to a monitored phenomenon. Also, a large number of sensor nodes may be involved in the observation of a single phenomenon. A changing set of sensor nodes may be involved in monitoring a single phenomenon over time. Raw sensor readings are often (pre-) processed inside the sensor network.

Despite these subtle differences, the transition between ubicomp and WSN is rather smooth, which will also be reflected by the design space we propose later in this chapter.

2.1.3 Peer-to-Peer Systems

Peer-to-peer (P2P) systems can be defined as self-organizing, decentralized distributed systems where nodes have symmetric roles. While originally conceived for file sharing in the Internet, the scalability and resilience of P2P systems lends itself to a growing domain of applications. Most P2P research is concerned with the establishment of so-called *overlay networks* on top of existing communication infrastructures such as the Internet. Such techniques effectively shield application designers from the complexities of organizing and maintaining an overlay topology, of tolerating node failures, of balancing load, and of locating application objects.

P2P systems share a number of properties with wireless sensor networks, namely they are both large-scale, self-organizing, decentralized distributed systems. Hence, there is a high potential for P2P techniques in the sensor network context. However, there are also a number of differences between typical P2P systems and WSN, which make an adoption non-trivial. Most prominently, P2P systems are typically designed for wired networks, where nodes do not suffer from resource and energy constraints. Also, neighbor nodes in overlay networks often map to distant nodes in the underlying physical network. In the sensor network context, this may result in poor performance and high energy consumption.

2.1.4 Embedded Systems

Many artifacts of our daily life such as consumer electronics, other consumer products, and vehicles contain computers, which are commonly referred to as embedded systems. These embedded computing systems monitor and control certain aspects of the containing system. In contrast to general purpose computing systems, embedded system perform a single or tightly knit set of application-specific functions, are often subject to real-time constraints, and must meet stringent dependability requirements.

Embedded computing systems show a high level of diversity, ranging from centralized, highly cost/resource/energy-constrained systems (e.g., a pocket remote control), to high-performance distributed embedded systems (e.g., an aircraft autopilot). Most distributed embedded systems use wired communication, because it is less prone to communication errors and offers more deterministic behavior than wireless communication.

An embedded system consists of hardware and software components, where the allocation of functions to hardware and software is highly dependent on the requirements of the actual system. For performance and security reasons, part of the functionality may be provided by application-specific hardware components. The use of software in conjunction with programmable hardware provides flexibility and support for more complex features.

What differentiates wireless sensor networks from traditional embedded systems is the large number of participating nodes, the use of wireless communication, and often severe resource/cost/energy constraints of sensor nodes. Traditional embedded systems are typically a fixed part of highly engineered structures, that is, the actual "embedding" takes place at production time and does not change during the lifetime of the embedded system. In contrast, wireless sensor nodes may be deployed in a natural setting with little control over the actual placement and distribution of the nodes. Also, the embedding (i.e., deployment) of sensor nodes into the physical environment is not tied to the production time of the sensor node and may change during the lifetime of a sensor node.

2.1.5 Remote and Wired Sensing

Many existing systems for observing real-world phenomena are based on few sensors with a relatively long range, such as satellites for earth observation, weather stations, or sonars. Due to the long range, these systems can observe phenomena that are far away from the sensors. However, the resolution of these systems decreases with the observation distance. Moreover, many systems require a free line of sight between the sensors and the observed phenomenon.

Wireless sensor nodes are equipped with short range sensors. Many of these devices are placed in the close vicinity of the observed phenomenon. Since the average distance between the observed phenomenon and the sensors is small and since many redundant sensors observe a single phenomenon, the effective monitoring resolution of a wireless sensor network can be better than that of a remote sensing approach. The placement of the sensors close the phenomenon allows the use of wireless sensor networks also in cluttered environments where line-of-sight paths are rather short.

Many existing systems make use of distributed, wired sensors that are connected to a central computing device (e.g., sensors in cars and engines, sonar arrays). Such an approach has a number of advantages: sensors do not need separate power supplies, the wired network has a fixed topology, small and deterministic delays, communication errors are very rare. On the other hand, the wiring limits the flexibility and scale of such wired sensor networks.

2.1.6 Wireless, Mobile, and Ad Hoc Networks

Wireless communication, especially with focus on short communication range and low power consumption, is a key enabling technology for wireless sensor networks. In mobile networks, computers capable of wireless communication can change their physical position over time, resulting in dynamically changing network topologies. Ad hoc networks are wireless networks that do not require an external infrastructure such as base stations in mobile phone networks. The nodes of an ad hoc network act both as sources/sinks of messages and as routers that forward messages on behalf of other nodes. Nodes can join and leave the network anytime. Although ad hoc networks may also consist of immobile nodes, they often contain mobile nodes. Power awareness is an important issue in the context of mobile networks, since mobile computing devices are often powered by batteries. Recent research in mobile ad hoc networks focuses on routing, mobility management, power management, self-configuration, and the radio interface (including the radio hardware and medium access techniques).

It is anticipated that many wireless sensor networks will be implemented as a mobile ad hoc network (MANET). However, results from MANET research often cannot be directly applied to wireless sensor networks, since resource and energy constraints are typically more stringent here. Typical MANET research focuses on handheld devices or laptops with renewable batteries. The computing, storage, communication resources of these devices are comparable to desktop computers. In contrast, sensor node batteries are often not replaceable; range, bandwidth, reliability of wireless communication links, computing and memory resources, and available energy may be orders of magnitude smaller compared to more traditional MANET nodes.

Wireless sensor networks may also rely on infrastructure-based mobile networks. For example, mobile phone companies are currently exploring the value of mobile phones for sensor networks. Such networks could either solely consist of mobile phones equipped with sensors, or a mobile phone could act as a gateway connecting an ad hoc sensor network to the phone network. Such combinations of infrastructure-based and ad hoc networks would allow remote access to sensor networks and an integration with existing computing infrastructures.

2.1.7 Digital Signal Processing

Digital signal processing (DSP) can be defined as the analysis and modification of discrete time signals (i.e., sequences of numbers). It is a key technology for systems that process signals from the real world. While wireless sensor networks do process real-world signals, there are quite a number of differences between many traditional signal processing systems and sensor networks.

First of all, many traditional DSP application are centralized, that is, data from possibly many sources is collected at a single processor for evaluation. Depending on the bandwidth of the input signals, this approach may require high-bandwidth communication channels and a very powerful centralized processor.

In typical sensor networks, the effective channel bandwidth and processing power is typically rather limited. Hence, centralized processing of digital signals is often not desirable. Rather, input signals should be preprocessed locally on the originating sensor nodes to extract relevant features that can be communicated more efficiently. Hence, DSP in sensor networks is often decentralized and distributed over the sensor nodes.

2.2 The Sensor Network Design Space

In the recent past, wireless sensor networks have found their way into a wide variety of applications and systems with vastly varying requirements and characteristics. As a consequence, it is becoming increasingly difficult to discuss typical requirements regarding hardware issues and software support. This is particularly problematic in a multidisciplinary research area such as wireless sensor networks, where close collaboration between users, application domain experts, hardware designers, and software developers is needed to implement efficient systems.

Initial research into wireless sensor networks was mainly motivated by military applications, with DARPA continuing to fund a number of prominent research projects (e.g., Smart Dust, NEST) that are commonly regarded as the cradle of sensor-network research. The type of applications considered by these projects led to a de facto definition of a wireless sensor network as a large-scale (possibly thousands of nodes, covering large geographical areas), wireless, ad hoc, multi-hop, unpartitioned network of homogeneous, tiny (hardly noticeable), mostly immobile (after deployment) sensor nodes that would be randomly deployed in the area of interest.

More recently, other, civilian application domains of wireless sensor networks have been considered, such as environmental and species monitoring, agriculture, production and delivery, healthcare, etc. (see Section 2.4). Concrete projects targeting these application areas indicate that the above definition of a wireless sensor network does not necessarily apply for these applications – networks may consist of heterogeneous and mobile sensor nodes, the network topology may be as simple as a star topology, networks may make use of existing communication infrastructures, etc. To meet this general trend towards diversification, we will discuss important dimensions of the sensor network design space in the following subsections. We will informally characterize each of the dimensions and, where appropriate, identify (possibly orthogonal) property classes in order to support a coarse-grained classification of sensor network applications.

It is certainly debatable which issues are important enough to be explicitly considered as dimensions in the design space and one could argue in favor of adding more dimensions or removing some from our suggestions detailed below. In fact, we expect that this might become reasonable in the future as the field and its applications evolve. However, we have tried to ensure that our initial suggestion consisted of a sensible set of dimensions, by basing our choice on the following two principles. Firstly, there should be notable variability between applications with

respect to dimensions. Secondly, a dimension should have a significant impact on the design and implementation of technical solutions.

In the subsequent section we show that existing and envisioned applications of wireless sensor networks can indeed be located at different points in the design space, with a number of important implications.

2.2.1 Deployment

The deployment of sensor nodes in the physical environment may take several forms. Nodes may be deployed at random (e.g., by dropping them from an aircraft) or installed at deliberately chosen spots. Deployment may be a one-time activity, where the installation and use of a sensor network are strictly separate activities. However, deployment may also be a continuous process, with more nodes being deployed at any time during the use of the network – for example, to replace failed nodes or to improve coverage at certain interesting locations.

The actual type of deployment affects important properties such as the expected node density, node locations, regular patterns in node locations, and the expected degree of network dynamics.

We suggest the following coarse-grained classification with respect to deployment: *random vs. manual; one-time vs. iterative.*

2.2.2 Mobility

Sensor nodes may change their location after initial deployment. Mobility can result from environmental influences such as wind or water, sensor nodes may be attached to or carried by mobile entities, and sensor nodes may possess automotive capabilities. In other words, mobility may be either an incidental side effect, or it may be a desired property of the system (e.g., to move nodes to interesting physical locations), in which case mobility may be either active (i.e., automotive) or passive (e.g., attached to a moving object not under the control of the sensor node). Mobility may apply to all nodes within a network or only to subsets of nodes. The degree of mobility may also vary from occasional movement with long periods of immobility in between, to constant travel.

Mobility has a large impact on the expected degree of network dynamics and hence influences the design of networking protocols and distributed algorithms. The actual speed of movement may also have an impact, for example on the amount of time during which nodes stay within communication range of each other.

We suggest the following coarse-grained classification with respect to mobility: *immobile vs. partly vs. all; occasional vs. continuous; active vs. passive.*

2.2.3 Cost, Size, Resources, and Energy

Depending on the actual needs of the application, the form factor of a single sensor node may vary from the size of a shoe box (e.g., a weather station) to a microscopically small particle (e.g., for military applications where sensor nodes should be

almost invisible). Similarly, the cost of a single device may vary from hundreds of Euros (for networks of very few, but powerful nodes) to a few Cents (for large-scale networks made up of very simple nodes).

Since sensor nodes are untethered autonomous devices, their energy and other resources are limited by size and cost constraints. Varying size and cost constraints directly result in corresponding varying limits on the energy available (i.e., size, cost, and energy density of batteries or devices for energy scavenging), as well as on computing, storage, and communication resources. Hence, the energy and other resources available on a sensor node may also vary greatly from system to system. Power may be either stored (e.g., in batteries) or scavenged from the environment (e.g., by solar cells).

These resource constraints limit the complexity of the software executed on sensor nodes. For our classification, we have partitioned sensor nodes roughly into four classes based on their physical size: *brick vs. matchbox vs. grain vs. dust.*

2.2.4 Heterogeneity

Early sensor network visions anticipated that sensor networks would typically consist of homogeneous devices that were mostly identical from a hardware and software point of view. Some projects, such as Amorphous Computing [1], even assumed that sensor nodes were indistinguishable, that is, they did not even possess unique addresses or IDs within their hardware. This view was based on the observation that otherwise it would not be feasible to cheaply produce vast quantities of sensor nodes.

However, in many prototypical systems available today, sensor networks consist of a variety of different devices. Nodes may differ in the type and number of attached sensors; some computationally more powerful "compute" nodes may collect, process, and route sensory data from many more limited sensing nodes; some sensor nodes may be equipped with special hardware such as a GPS receiver to act as beacons for other nodes to infer their location; some nodes may act as gateways to long-range data communication networks (e.g., GSM networks, satellite networks, or the Internet).

The degree of heterogeneity in a sensor network is an important factor since it affects the complexity of the software executed on the sensor nodes and also the management of the whole system.

We suggest the following coarse-grained classification with respect to heterogeneity: *homogeneous vs. heterogeneous.*

2.2.5 Communication Modality

For wireless communication among sensor nodes, a number of communication modalities can be used such as radio, diffuse light, laser, inductive and capacitive coupling, or even sound.

The most common modality is radio waves, since these do not require a free

line of sight, and communication over medium ranges can be implemented with relatively low power consumption and relatively small antennas (a few centimeters in the common sub-GHz frequency bands). Using light beams for communication requires a free line of sight and may interfere with ambient light and daylight, but allows for much smaller and more energy-efficient transceivers compared to radio communication. Smart Dust [49], for example, uses laser beams for communication. Inductive and capacitive coupling only works over small distances, but may be used to power a sensor node. Most passive Radio Frequency Identification (RFID) systems use inductive coupling, for example. Sound or ultrasound is typically used for communication under water or to estimate distances based on time-of-flight measurements. Sometimes, multiple modalities are used by a single sensor network system.

The communication modality used obviously influences the design of medium access protocols and communication protocols, but also affects other properties that are relevant to the application.

We suggest the following coarse-grained classification with respect to communication modality: *radio vs. light vs. inductive vs. capacitive vs. sound.*

2.2.6 Infrastructure

The various communication modalities can be used in different ways to construct an actual communication network. Two common forms are so-called infrastructure-based networks on the one hand and ad hoc networks on the other hand. In infrastructure-based networks, sensor nodes can only directly communicate with so-called base station devices. Communication between sensor nodes is relayed via the base station. If there are multiple base stations, these have to be able to communicate with each other. The number of base stations depends on the communication range and the area covered by the sensor nodes. Mobile phone networks and Smart Dust [49] are examples of this type of network.

In ad hoc networks, nodes can directly communicate with each other without an infrastructure. Nodes may act as routers, forwarding messages over multiple hops on behalf of other nodes.

Since the deployment of an infrastructure is a costly process, and the installation of an infrastructure may often not be feasible, ad hoc networks are preferred for many applications. However, if an infrastructure is already available anyway (such as the GSM network), it might also be used for certain sensor network applications.

Combinations of ad hoc networks and infrastructure-based networks are sometimes used, where clusters of sensor nodes are interconnected by a wide area infrastructure-based network.

Note that the above arguments not only apply to communication, but also to other infrastructures, such as localization or time synchronization (e.g., GPS satellites).

We suggest the following coarse-grained classification with respect to infrastructure: *infrastructure vs. ad hoc.*

2.2.7 Network Topology

One important property of a sensor network is its diameter, that is, the maximum number of hops between any two nodes in the network. In its simplest form, a sensor network forms a single-hop network, with every sensor node being able to directly communicate with every other node. An infrastructure-based network with a single base station forms a star network with a diameter of two. A multi-hop network may form an arbitrary graph, but often an overlay network with a simpler structure is constructed such as a tree or a set of connected stars.

The topology affects many network characteristics such as latency, robustness, and capacity. The complexity of data routing and processing also depends on the topology.

We suggest the following coarse-grained classification with respect to network topology: *single-hop vs. star vs. networked stars vs. tree vs. graph.*

2.2.8 Coverage

The effective range of the sensors attached to a sensor node defines the coverage area of a sensor node. Network coverage measures the degree of coverage of the area of interest by sensor nodes. With sparse coverage, only parts of the area of interest are covered by the sensor nodes. With dense coverage, the area of interest is completely (or almost completely) covered by sensors. With redundant coverage, multiple sensors cover the same physical location. The actual degree of coverage is mainly determined by the observation accuracy and redundancy required. Coverage may vary across the network. For example, nodes may be deployed more densely at interesting physical locations.

The degree of coverage also influences information-processing algorithms. High coverage is a key to robust systems and may be exploited to extend the network lifetime by switching redundant nodes to power-saving sleep modes.

We suggest the following coarse-grained classification with respect to coverage: *sparse vs. dense vs. redundant.*

2.2.9 Connectivity

The communication ranges and physical locations of individual sensor nodes define the connectivity of a network. If there is always a network connection (possibly over multiple hops) between any two nodes, the network is said to be connected. Connectivity is intermittent if the network may be occasionally partitioned. If nodes are isolated most of the time and enter the communication range of other nodes only occasionally, we say that communication is sporadic. Note that despite the existence of partitions, messages may be transported across partitions by mobile nodes.

Connectivity mainly influences the design of communication protocols and methods of data gathering.

We suggest the following coarse-grained classification with respect to connectivity: *connected vs. intermittent vs. sporadic.*

2.2.10 Network Size

The number of nodes participating in a sensor network is mainly determined by requirements relating to network connectivity and coverage, and by the size of the area of interest. The network size may vary from a few nodes to thousands of sensor nodes or even more. The network size determines the scalability requirements with regard to protocols and algorithms.

2.2.11 Lifetime

Depending on the application, the required lifetime of a sensor network may range from some hours to several years. The necessary lifetime has a high impact on the required degree of energy efficiency and robustness of the nodes.

2.2.12 Other QoS Requirements

Depending on the application, a sensor network must support certain quality-of-service aspects such as real-time constraints (e.g., a physical event must be reported within a certain period of time), robustness (i.e., the network should remain operational even if certain well-defined failures occur), tamper-resistance (i.e., the network should remain operational even when subject to deliberate attacks), eavesdropping-resistance (i.e., external entities cannot eavesdrop on data traffic), unobtrusiveness or stealth (i.e., the presence of the network must be hard to detect). These requirements may impact on other dimensions of the design space such as coverage and resources.

2.3 Implications of the Design Space

There are several important consequences of the design space as discussed above. Clearly, a single hardware platform will most likely not be sufficient to support the wide range of possible applications (cf. Section 2.5). In order to avoid the development of application-specific hardware, it would be desirable, however, to have available a (small) set of platforms with different capabilities that cover the design space. A modular approach, where the individual components of a sensor node can be easily exchanged, might help to partially overcome this difficulty. Principles and tools for selecting suitable hardware components for particular applications would also be desirable.

As similar observation can be made regarding algorithms and software in general. As with hardware, one could try to cover the design space with a (larger) set of different protocols, algorithms, and basic services. Note that this does in particular apply to algorithms for locating sensor nodes in space and time as will

be discussed in Chapter 3. The selection of appropriate software components for the application at hand and perhaps also dynamic adaptation during runtime of an application should be supported by appropriate frameworks.

In traditional distributed systems, middleware has been introduced to hide such complexity from the software developer by providing programming abstractions that are applicable for a large class of applications. This raises the question, whether appropriate abstractions and middleware concepts can be devised that are applicable for a large portion of the sensor network design space. This is not an easy task, since some dimensions of the design space (e.g., network connectivity) are very hard to hide from the system developer. Moreover, exposing certain application characteristics to the system and vice versa is a key approach for achieving energy and resource efficiency in sensor networks. Even if the provision of abstraction layers is conceptually possible, they often introduce significant resource overheads – which is problematic in highly resource-constrained environments such as sensor networks.

In addition to these more technical issues, the design space we advocate can hopefully bring more clarity to the often somewhat diffuse discussions about *typical* or *right* characteristics and requirements of wireless sensor networks.

2.4 Applications

Wireless sensor networks can be considered as a tool for observing real-world processes. In particular, the use of WSN might be a worthwhile option for observation tasks with one or more of the following properties:

- The observation environment is cluttered and can be hardly observed from afar.
- Any instrumentation for observation must be unobtrusive to avoid influencing observation results.
- The phenomenon of interest or its close physical environment can be instrumented for observation.
- A high spatial and temporal monitoring resolution is required.
- The signal-to-noise ratio of signals emitted by the phenomenon of interest is low or decreases significantly over distance.
- Traditional observation methods are very costly due to the involvement of human personnel.
- The observation environment is very harsh, inaccessible, or even toxic.
- The observation must be continuously performed during long periods of time or over large geographical areas.

The above problem characterization applies to a wide range of problem domains some of which we will sketch in the following sections. For each of these domains we describe some representative applications. We also show that these applications can be located at different points in the design space. In addition to these concrete applications, we briefly sketch application ideas and visions that have not yet been realized or which are not well-documented.

2.4.1 Species Monitoring

Wireless sensor networks can be used to observe the behavior of animals in their natural habitats. Cerpa et al [22] give an overview and motivation of this application domain.

Bird Observation at Great Duck Island

A WSN is being used to observe the breeding behavior of a small bird called Leach's Storm Petrel [58] on Great Duck Island, Maine, USA. These birds are easily disturbed by the presence of humans, hence WSN seems an appropriate way of better understanding their behavior. The breeding season lasts for seven months from April to October. The biologists are interested in the usage pattern of their nesting burrows, changes in environmental conditions outside and inside the burrows during the breeding season, variations among breeding sites, and the parameters of preferred breeding sites.

Sensor nodes are installed inside the burrows and on the surface. Nodes can measure humidity, pressure, temperature, and ambient light level. Burrow nodes are equipped with infrared sensors to detect the presence of the birds. The burrows occur in clusters and the sensor nodes form a multi-hop ad hoc network. Each network cluster contains a sensor node with a long-range directional antenna that connects the cluster to a central base station computer. The base station computer is connected to a database back-end system via a satellite link. Sensor nodes sample their sensors about once a minute and send their readings directly to the database back-end system.

Deployment	manual, one-time
Mobility	immobile
Resources	matchbox
Cost	approx. 200 USD per node
Energy	battery, solar panel
Heterogeneity	weather stations, burrow nodes, gateways
Modality	radio
Infrastructure	base station, gateways
Topology	star of subgraphs
Coverage	dense (every burrow)
Connectivity	connected

Size	tens to hundreds (about 100 deployed)
Lifetime	7 months (breeding period)

ZebraNet

A WSN is being used to observe the behavior of wild animals within a spacious habitat (e.g., wild horses, zebras, and lions) [48] at the Mpala Research Center in Kenya. Of particular interest is the behavior of individual animals (e.g., activity patterns of grazing, graze-walking, and fast moving), interactions within a species, interactions among different species (e.g., grouping behavior and group structure), and the impact of human development on the species. The observation period is scheduled to last a year or more. The observation area may be as large as hundreds or even thousands of square kilometers.

Animals are equipped with sensor nodes. An integrated GPS receiver is used to obtain estimates of their position and speed of movement. Light sensors are used to give an indication of the current environment. Further sensors (head up or down, body temperature, ambient temperature) are planned for the future. Each node logs readings from its sensors every three minutes. Whenever a node enters the communication range of another node, the sensor readings and the identities of the sensor nodes are exchanged (i.e., data is flooded across network partitions). At regular intervals, a mobile base station (e.g., a car or a plane) moves through the observation area and collects the recorded data from the animals it passes.

Deployment	manual, one-time
Mobility	all, continuous, passive
Resources	matchbox
Energy	battery
Heterogeneity	nodes, gateway
Modality	radio
Infrastructure	base station, GPS
Topology	graph
Coverage	dense (every animal)
Connectivity	sporadic
Size	tens to hundreds
Lifetime	one year

Further Applications in Species Monitoring

A variety of different sensor devices are used to monitor the behavior of Whale Sharks [137] around the Seychelles. The so-called archival pop-off tag is attached to the shark and uses light and pressure sensors to monitor the diving behavior. After a certain time, the tag detaches from the shark and floats to the surface, from where it sends off collected data to a satellite.

Sensor devices are also used to observe social interactions among seals [127] in

the Northern Sea. A sensor node [37] is attached to each seal which can detect the distance to and the identity of another nearby tag (and seal). These sightings are collected and sent off to a satellite.

A project at CENS intends to use a sensor network for detection and classification of marine microorganisms (e.g., alga) that are toxic to marine life and dangerous to human health [123]. Work in this project has so far focused on the development of appropriate sensors. Obviously, these sensors cannot be attached to microorganisms, but have to be installed in an area of interest in the ocean.

2.4.2 Environmental Monitoring

Besides animals, a number of other environmental phenomena can be observed with wireless sensor networks.

Glacier Monitoring

A sensor network is being used to monitor sub-glacier environments at Briksdalsbreen, Norway, with the overall goal of better understanding the Earth's climate [62]. Of particular interest are displacements and the dynamics inside the glacier. A lengthy observation period of months to years is required.

Sensor nodes are deployed in drill holes at different depths in the glacier ice and in the till beneath the glacier. Sensor nodes are equipped with pressure and temperature sensors and a tilt sensor for measuring the orientation of the node. Sensor nodes communicate with a base station deployed on top of the glacier. The base station measures supra-glacial displacements using differential GPS and transmits the data collected via GSM. Nodes are not recoverable after deployment. Radio communication through ice and water is a major problem.

Deployment	manual, one-time
Mobility	all, continuous, passive
Resources	brick
Energy	battery
Heterogeneity	nodes, base station
Modality	radio, GSM
Infrastructure	base station, GPS, GSM
Topology	star
Coverage	sparse
Connectivity	connected
Size	9 deployed (potential for tens to hundreds)
Lifetime	several months

Bathymetry

A sensor network is being used to monitor the impact on the surrounding environment of a wind farm off the coast of England [61]. Of particular interest here is

the influence on the structure of the ocean bed (e.g., formation of sand banks) and the influence on tidal activity.

Sensor nodes are deployed on the ocean bed by dropping them from a ship at selected positions, their location being fixed on the ocean bed by an anchor. Each sensor node is connected via a cable to a buoy on the ocean surface that contains the radio equipment and GPS, since radio communication under water is virtually impossible. The sensor nodes are able to measure pressure, temperature, conductivity, current, and turbidity, and form a self-organized ad hoc network.

Deployment	manual, one-time
Mobility	all, occasional, passive
Resources	brick
Energy	battery
Heterogeneity	homogeneous
Modality	radio
Infrastructure	GPS
Topology	graph
Coverage	sparse (500m - 1km apart)
Connectivity	connected
Size	6 deployed, 50 planned (potential for up to hundreds)
Lifetime	several months

Ocean Water Monitoring

The ARGO project [112] is using a sensor network to observe the temperature, salinity, and current profile of the upper ocean. The goal is a quantitative description of the state of the upper ocean and the patterns of ocean climate variability, including heat and freshwater storage and transport. Intended coverage is global, and observation is planned to last for several years. Measurement data is available almost in real-time.

The project uses free-drifting profiling sensor nodes equipped with temperature and salinity sensors. The nodes are dropped from ships or planes. The nodes cycle to a depth of 2000m every ten days. Data collected during these cycles is transmitted to a satellite while nodes are at the surface. The lifetime of the nodes is about 4-5 years.

Deployment	random, iterative
Mobility	all, continuous, passive
Resources	brick
Cost	approx. 15000 USD per node
Energy	battery
Heterogeneity	homogeneous
Modality	radio
Infrastructure	satellite

Topology	star
Coverage	sparse
Connectivity	intermittent
Size	1300 deployed (3000 planned)
Lifetime	4-5 years

Further Applications in Environmental Monitoring

Several projects explore the use of sensor networks for monitoring water quality. The European project SEWING [129] is currently concerned with the development of sensors for certain chemicals. It is, however, anticipated to construct complete sensor nodes for water quality monitoring. At CENS, a similar project intends to observe water quality and to monitor transport of contaminants with ground water using sensor networks [118].

In [107], as wireless sensor network for monitoring volcanic eruptions is presented. The system consists of several infrasound monitoring nodes, which report low-frequency acoustic signals to an aggregator node, which preprocesses the data and sends aggregated values to a remote base station via a long-range radio link. A GPS node is used to synchronize the infrasound monitoring nodes. The system can be used to monitor and locate volcanic eruptions.

Sensor networks can be used to monitor seismic activity and the structural response of buildings [51]. A number of experiments with seismic sensor networks in buildings have been carried out within CENS [109, 128]. A sensor network is also used to monitor the influence of winds on Golden Gate Bridge in San Francisco [133].

Another application of sensor networks is the observation of micro climates and their changes over time [27, 122]. One particularly interesting application in this context is the use of sensor networks for detecting signs of life on other planets (e.g., on Mars) [28].

2.4.3 Agriculture

Wireless sensor networks can also be used to increase the efficiency of plant breeding and livestock husbandry.

Grape Monitoring

A WSN is being used to monitor the conditions that influence plant growth (e.g., temperature, soil moisture, light, and humidity) across a large vineyard in Oregon, USA [11]. The goals include supporting precision harvesting (harvesting an area as soon as the grapes in it are ripe), precision plant care (adapting the water/fertilizer/pesticide supply to the needs of individual plants), frost protection, predicting insect/pest/fungi development, and developing new agricultural models.

In a first version of the system, sensor nodes are deployed across a vineyard in a regular grid about 20 meters apart. A temperature sensor is connected to each sensor node via a cable in order to minimize false sensor readings due to heat disseminated by the sensor nodes. A laptop computer is connected to the sensor network via a gateway to display and log the temperature distribution across the vineyard. The sensor nodes form a two-tier multi-hop network, with nodes in the second tier sending data to a node in the first tier. Nodes in the first tier also collect sensor data, but do additionally act as data routers.

Deployment	manual, one-time
Mobility	immobile
Resources	matchbox
Cost	200 USD per node
Energy	battery
Heterogeneity	sensors, gateway, base station
Modality	radio
Infrastructure	base station
Topology	tree (two-tiered multi-hop)
Coverage	sparse (20m apart)
Connectivity	connected
Size	65 deployed (potential for up to hundreds)
Lifetime	several months (growth period)

Cattle Herding

A WSN is being used to implement virtual fences, with an acoustic stimulus being given to animals that cross a virtual fence line [18]. Movement data from the cows controls the virtual fence algorithm that dynamically shifts fence lines. Such a system can reduce the overheads of installing and moving physical fences and can improve the usage of feedlots.

For the first experiment, each sensor node consists of a PDA with a GPS receiver, a WLAN card, and a loudspeaker for providing acoustic stimuli to the cattle as they approach a fence. These devices are attached to the neck of the cows. The nodes form a multi-hop ad hoc network, forwarding movement data to a base station (a laptop computer). The base station transmits fence coordinates to the nodes.

Deployment	manual, one-time
Mobility	all, continuous, passive
Resources	brick
Cost	approx. 1000 USD per node
Energy	battery
Heterogeneity	homogeneous
Modality	radio
Infrastructure	base station, GPS

Topology	graph
Coverage	dense (every cow)
Connectivity	intermittent
Size	10 deployed (potential for up to hundreds)
Lifetime	days to weeks

Further Applications in Agriculture

The Hogthrob project [135] intends to use sensor networks for sow monitoring. In particular, movement data is intended to be used to detect the sow's heat period and to detect abnormal behavior which could be caused by diseases.

The PlantCare project [53] uses a sensor network to monitor soil humidity of plants and to control a robot to water indigent plants.

2.4.4 Production and Delivery

In this section we consider the use of wireless sensor networks for monitoring the production and delivery of goods.

Cold Chain Management

The commercial Securifood system [78] is a WSN for monitoring the temperature compliance of cold chains from production, via distribution centers and stores, to the consumer. Clients receive an early warning of possible breaks in the cold chain.

The system consists of four major components: sensor nodes, relay units, access boxes, and a warehouse. Sensor nodes are transported with the products and collect temperature data. Relay units collect and store temperature data from sensor nodes – they are more powerful devices with a permanent power supply. Multiple relay units form a multi-hop ad hoc network. An access box is an even more powerful embedded Linux device that acts as a gateway between the network of relay units and the Internet. There is one access box per production site. An Internet-hosted data warehouse acts as a central server, collecting data from all the access boxes. The data warehouse provides an online image of all the sensor data in the system and acts as a central data repository for applications.

Deployment	manual, iterative
Mobility	partly (sensors), occasional, passive
Resources	matchbox (sensors), brick (relays)
Energy	battery
Heterogeneity	sensor units, relay units, access boxes, warehouse
Modality	radio
Infrastructure	relays, access boxes
Topology	tree (three-tiered multi-hop)
Coverage	sparse

Connectivity	intermittent
Size	55 sensor units and 4 relays deployed (potential for hundreds)
Lifetime	years

Further Applications in Production and Delivery

Wireless sensor networks can also be used to monitor and manage the life cycle of production tools [9]. This applies in particular to tools that wear off and need regular treatment to avoid failures. Additionally, sensor network technology can help improve the availability of mobile tools and equipment. Intel research, for example, is examining the use of vibration sensors for early detection of problems with the cooling equipment in a semiconductor fabrication unit [136]. A similar approach could also be used to detect potential problems in oil wells and pipelines [76]. A number of ongoing projects at BP [16] use sensor networks for monitoring ship vibrations and pipeline integrity.

2.4.5 Disaster Relief

Wireless sensor networks can be used in emergency situations, for example, to coordinate and increase the efficiency of rescue actions.

Rescue of Avalanche Victims

A WSN is being used to assist rescue teams in saving people buried in avalanches [67]. The goal is to better locate buried people and to limit overall damage by giving the rescue team additional indications of the state of the victims and to automate the prioritization of victims (e.g., based on heart rate, respiration activity, and level of consciousness).

For this purpose, people at risk (e.g., skiers, snowboarders, and hikers) carry a sensor node that is equipped with an oximeter (a sensor which measures the oxygen level in blood), and which permits heart rate and respiration activity to be measured. Additionally, an oxygen sensor is used to detect air pockets around the victim. Accelerometers are used to derive the orientation of the victim. The rescue team uses a PDA to receive sensory data from the buried victims.

Deployment	manual, one-time
Mobility	all, continuous, passive
Resources	matchbox
Energy	battery
Heterogeneity	homogeneous
Modality	radio
Infrastructure	rescuer's PDA
Topology	star
Coverage	dense (every person)

Connectivity	connected
Size	tens to hundreds (number of victims)
Lifetime	days (duration of a hike)
QoS	dependability

Further Applications in Disaster Relief

In flood situations, dams of sank sacks are often used to protect against water. One common problem with this approach is that sand sacks get wet, which may eventually lead to water inleakage and to a collapse of the dam. By equipping sand sacks with sensor nodes, such situations can be detected early on and people can be guided to the defective place in order to fix them before bad things happen. (This idea goes back to researchers at the University of Rostock.)

Sensor networks may be used to assist firefighters in defeating large scale forest fires and can help protect the lives of firefighters [52]. A sensor network may measure wind direction and speed to help predict direction and speed of spreading of the fire. For this, a sensor network may be deployed in affected areas during firefighting.

2.4.6 Building Management and Automation

Modern buildings do already contain a large number of wired sensors and actuators to control a variety of functions (e.g., heat control, door openers, automatic light and blind control) [126]. It has been argued that replacing these wired sensor by wireless sensor networks could reduce construction cost and increase flexibility by removing the cabling. However, wireless sensor networks may also enable a number of novel applications in this context as indicated below.

Power Monitoring

A WSN is being used to monitor power consumption in large and dispersed office buildings [50]. The goal is to detect locations or devices that are consuming a lot of power to provide indications for potential reductions in power consumption.

The system consists of three major components: sensor nodes, transceivers, and a central unit. Sensor nodes are connected to the power grid (at outlets or fuse boxes) to measure power consumption and for their own power supply. Sensor nodes directly transmit sensor readings to transceivers. The transceivers form a multi-hop network and forward messages to the central unit. The central unit acts as a gateway to the Internet and forwards sensor data to a database system.

Deployment	manual, iterative
Mobility	immobile
Resources	matchbox
Energy	power grid

Heterogeneity	sensor nodes, transceivers, central unit
Modality	radio (sensors unidirectional)
Infrastructure	transceivers
Topology	layered multi-hop
Coverage	sparse (selected outlets)
Connectivity	connected
Size	tens to hundreds
Lifetime	years (building lifecycle)

Further Applications in Building Management and Automation

In [87], a wireless sensor network for commercial lighting control is suggested. For this, sensor nodes are deployed in a building to sense the occupancy status of individual rooms. The obtained raw data is fed to a decision system that controls the lights.

It was also suggested to use sensor networks for calibration of air conditioning systems to the particular installation environments to reduce the noise level and power consumption of these systems [46]. For this purpose, a portable sensor network is installed in the room, providing temperature, humidity, and noise level distribution. These data can be used to fine tune the configuration of the air conditioning.

2.4.7 Traffic and Infrastructure

There is a growing trend to instrument cars with more and more sensors and actuators to improve the drivability and comfort. While past research mostly focused on single cars, recent developments include networking of cars [117, 119] to reduce accidents, traffic jams, environmental stress, or to improve fleet management.

Networked Parking Spaces

A sensor network is used to find free parking lots [8]. The system can help find streets in the locality with vacant spots, can find occupied parking meters within a certain range which will expire at a certain time, and can locate all vehicles that reside in expired spots.

In this system, parking meters are equipped with sensor nodes. These nodes are equipped with sensors to detect the occupancy status of the according parking spot and have access to parameters of the parking meter such as time of expiry. The sensor nodes form a static multi-hop ad hoc network. Cars are also equipped with sensor nodes that establish a link to the meter network to issue queries about free parking spots.

Deployment	manual
Mobility	partly (car nodes)

Resources	matchbox
Energy	car power system, solar, battery
Heterogeneity	car nodes, meter nodes
Modality	radio
Infrastructure	ad hoc
Topology	multi-hop
Coverage	dense
Connectivity	intermittent (car - meter network)
Size	hundreds to thousands
Lifetime	years (lifecycle of meters/cars)
QoS	robustness

Further Applications in Traffic and Infrastructure

Sensor nodes installed alongside roads can serve a number of purposes, among others to improve safety, to improve traffic flow, and to improve environmental health [72]. Based on local environmental data (e.g., road surface temperature), such sensor devices can issue warnings or even control vehicle speed. Roadside sensor nodes can also be used for traffic monitoring and can thus help predict traffic flow and jams. For example, in [24] a wireless sensor network is presented for vehicle detection, estimation of vehicle length, and speed measurements.

2.4.8 Home and Office

Sensor networks can also improve the convenience of home and office environments.

Furniture Assembly

A WSN is being used to assist people during the assembly of complex composite objects such as do-it-yourself furniture [3]. This saves users from having to study and understand complex instruction manuals, and prevents them from making mistakes.

The furniture parts and tools are equipped with sensor nodes. These nodes possess a variety of different sensors: force sensors (for joints), gyroscope (for screwdrivers), and accelerometers (for hammers). The sensor nodes form an ad hoc network for detecting certain actions and sequences thereof and give visual feedback to the user via LEDs integrated into the furniture parts.

Deployment	manual, one-time
Mobility	all, occasional, passive
Resources	matchbox
Energy	battery
Cost	approx. 100 Euro per node
Heterogeneity	different sensors

Modality	radio
Infrastructure	ad hoc
Topology	star
Coverage	sparse
Connectivity	connected
Size	tens
Lifetime	hours (duration of assembly)

Further Applications in Home and Office

In an office environment, sensor nodes can be attached to a number of artifacts such as coffee mugs [12] or chairs [13] in order to improve the efficiency and convenience of the environment. Sensor in chairs, for example, can be used to detect whether a chair is occupied or not. The status of multiple chairs in a room could be used to derive the occupancy status of the room, which could be displayed at an electronic doorplate (e.g., to keep people from entering the room), or could be sent to a central room management system (e.g., to improve room allocation).

2.4.9 Military and Homeland Security

The military and its funding agencies have been one of the main driving forces behind wireless sensor network research. More recently, the use of wireless sensor networks has also been considered for improving homeland security.

Vehicle Tracking

A WSN is being used to track the path of military vehicles (e.g., tanks) [134]. The sensor network should be unnoticeable and difficult to destroy. Tracking results should be reported within given deadlines.

Sensor nodes are deployed from an unmanned aerial vehicle (UAV). Magnetometer sensors are attached to the nodes in order to detect the proximity of tanks. Nodes collaborate in estimating the path and velocity of a tracked vehicle. Tracking results are transmitted to the unmanned aerial vehicle.

Deployment	random (thrown from aircraft)
Mobility	all, occasional, passive
Resources	matchbox
Cost	approx. 200 USD per node
Energy	battery
Heterogeneity	homogeneous
Modality	radio
Infrastructure	UAV
Topology	graph
Coverage	sparse

Connectivity	intermittent (UAV)
Size	5 deployed (potential for tens to thousands)
Lifetime	weeks to years (conflict duration)
QoS	stealth, tamper-resistance, real-time

Self-Healing Mine Field

Anti-tank landmines are being equipped with sensing and communication capabilities to ensure that a particular area remains covered even if the enemy tampers with a mine to create a potential breach lane [66]. If tampering is detected by the mine network, an intact mine hops into the breach using a rocket thruster.

The mines form a multi-hop ad hoc network and monitor radio link quality to detect failed mines. Nodes also estimate their location and orientation using ultrasonic ranging. When a node failure is detected, one of the mines is selected to relocate itself using one of eight rocket thrusters.

Deployment	manual
Mobility	all, occasional, active
Resources	brick
Energy	battery
Heterogeneity	homogeneous
Modality	radio, ultrasound (for localization)
Infrastructure	ad hoc
Topology	graph
Coverage	dense
Connectivity	connected
Size	20 deployed (potential for up to hundreds)
Lifetime	months to years
QoS	tamper-resistance

Further Applications in Military and Homeland Security

A number of research and development efforts are dedicated to sensor networks for detection, classification, and tracking of hostile activities (e.g., biological/chemical/radiological attacks, troops, tanks, vessels) in the military context. The goal of the DARPA-funded project ARGUS [113] was the development of advanced remote ground unattended sensors the would be dropped from aircraft to detect seismic and acoustic signatures and send them to a satellite. The later project MIUGS [100] focused on the same type of application, but sensors would form multi-hop ad hoc networks without using satellite communication. A number of DARPA-funded research projects such as NEST also work on energy-efficient sensor networks for target tracking [45].

One particularly noticeable effort in this context is the Seaweb project [77], which tries to accomplish near-real-time data telemetry for a set of widely spaced

oceanographic sensors. Sensor nodes are deployed on the ground of ocean/river/bay environments and communicate with each other via undersea acoustic signaling (telesonar). Such a system would be useful for the detection, classification, and tracking of vessels and mines, but also for the detection of biological, chemical, and radiological contamination.

2.4.10 Surveillance and Law Enforcement

Being an ideal tool for unobtrusive surveillance, sensor networks may also be used by executive authorities to enforce laws, to prevent and to elucidate offenses.

Sniper Localization

A WSN is being used to locate snipers and the trajectory of bullets [97], providing valuable clues for law enforcement. The system consists of sensor nodes that measure the muzzle blast and shock wave using acoustic sensors. The sensor nodes form a multi-hop ad hoc network. By comparing the time of arrival at distributed sensor nodes, the sniper can be localized with an accuracy of about one meter, and with a latency of under two seconds. The sensor nodes use an FPGA chip to carry out the complex signal processing functions.

Deployment	manual
Mobility	immobile
Resources	matchbox with FPGA
Cost	approx. 200 USD per node
Energy	battery
Heterogeneity	homogeneous
Modality	radio
Infrastructure	ad hoc
Topology	graph
Coverage	redundant (multiple nodes must recognize a shot)
Connectivity	connected
Size	60 deployed (potential for up to hundreds)
Lifetime	months to years
QoS	real-time

Further Applications in Surveillance and Law Enforcement

Sensor networks might also be helpful for surveillance of transient events such as construction sites, carnivals, crime scene surveillance, or temporary cubicle monitoring [4]. In these settings, existing surveillance systems often cannot be used due to their high cost and due to the deployment overhead. Sensor networks might also be helpful in the surveillance of widespread areas (e.g., border protection), where the use of traditional equipment would be too expensive or otherwise disadvantageous (e.g., cluttered environments).

2.4.11 Health Care

Networks of wireless sensors can be used to observe the state of health of humans. A discussion of this application domain can be found in [10, 92].

Vital Sign Monitoring

Wireless sensors are being used to monitor vital signs of patients in a hospital environment [6]. Compared to conventional approaches, solutions based on wireless sensors are intended to improve monitoring accuracy whilst also being more convenient for patients.

The system consists of four components: a patient identifier, medical sensors, a display device, and a setup pen. The patient identifier is a special sensor node containing patient data (e.g., name) which is attached to the patient when he or she enters the hospital. Various medical sensors (e.g., electrocardiogram) may be subsequently attached to the patient. Patient data and vital signs may be inspected using a display device. The setup pen is carried by medical personnel to establish and remove associations between the various devices. The pen emits a unique ID via infrared to limit the scope to a single patient. Devices which receive this ID form a body area network.

Deployment	manual
Mobility	all, continuous, passive
Resources	matchbox
Energy	battery
Heterogeneity	medical sensors, patient identifier, display device, setup pen
Modality	radio, IR light (for setup pen)
Infrastructure	ad hoc
Topology	single-hop
Coverage	dense
Connectivity	connected
Size	tens
Lifetime	days to months (hospital stay)
QoS	real-time, dependability, eavesdropping-resistance

Further Applications in Health Care

Body-worn sensor networks can also be used for automated detection and classification of activities (e.g., running, walking, standing, climbing stairs) and clinical symptoms (e.g., stress, epileptic seizures) [99]. These basic classifiers can be used to implement proactive healthcare, for example by assessing the healthiness of a lifestyle or by detecting diseases early on.

Sensor networks can also be used to assess social interactions [32], for example to detect and monitor physical and cognitive decline of elderly people [121].

2.5 Sensor Node Prototypes

The previous section pointed out that different applications require different types of sensor nodes and even heterogeneous networks consisting of different classes of sensor nodes. Below we discuss four prominent classes of devices and instances thereof.

2.5.1 Motes

The most commonly used class of sensor nodes at the time of writing is often referred to as *Motes*. These devices are built from commercially available general purpose electronic circuits. Including a battery, the size of a typical Mote is comparable to a matchbox. The most prominent example of this device class are the Berkeley Motes [115].

The major components of a Mote are an embedded microcontroller, a radio transceiver and antenna, interfaces to sensors and actuators, a real-time clock, circuitry for power conversion, and a battery. Some designs include additional memory or co-processors for speeding up computations. We will investigate these components in the following sections.

Microcontroller and Memory

Embedded microcontrollers such as the ATMEL AT Mega series used in many designs do not only provide a processor core, but program memory, general purpose memory, and a variety of input/output interfaces. The latter include Universal Asynchronous Receiver/Transmitters (UART) for serial IO, analog-to-digital converters, analog voltage comparators, a large number of freely programmable digital input and output signals, and various standard digital IO interfaces such as I2C and SPI, which are directly supported by many digital sensors. Additional general purpose memory is often provided by a separate chip. The exact capabilities of microcontrollers may vary, but typical numbers are a few MIPS processing power, few hundred kilobytes of program memory, few kilobytes of general purpose memory. External memory chips provide additional general purpose memory of tens to hundreds of kilobytes.

Radio and Antenna

The radio transceiver enables wireless networking of the Motes and is perhaps the component that varies most among different Mote designs. The most simple radios support a single communication channel and a simple on-off modulation scheme (e.g., RFM TR1000 used by early Berkeley Motes). The functionality of such radios is limited to a simple conversion between a digital signal and a modulated radio wave. There is no support for medium access control mechanisms such as handling of collisions. Communication is broadcast, that is, all receivers within communication range receive the signal. More elaborate radios such as the ChipCon (used

by more recent Berkeley Mote designs) support multiple communication channels and more robust modulation schemes. This type of radio typically has a shared bandwidth of some tens of kilobits per second and a communication range of few tens of meters.

Another class of radio are so-called radio modems. These are not simply radio transceivers on the physical layer, but provide much more sophisticated functionality (including MAC functions). Overall, these devices are somewhat similar to traditional modems, in that they accept complex commands such as "connect to X" or "send a packet to X". To support this functionality, they often include a separate microcontroller. One example of this type of radio is Bluetooth (used by the BTnode [116]). Bluetooth is connection-based, that is, prior to communication a connection has to be established between a pair of nodes. A broadcast mode is also supported, where a message is sent to all connected peers. The basic element of a Bluetooth network is the Piconet, which consists of a single master and up to seven slaves that form a star topology. Multiple Piconets can be connected to form multi-hop Scatternets. Bluetooth supports communication ranges of 10 or 100 meters and a shared bandwidth of one Megabit per second. A frequency hopping technique is used to minimize interference among Piconets and with other radio signals. While Bluetooth offers a number of advantages (e.g., simple to use, high bandwidth, standardized), it suffers from two major drawbacks: the energy consumption is rather high compared to the simple radios described above, and connection setup may take up to several seconds.

Some sensor node designs even support multiple radio frontends. Older designs included multiple identical radios in order to enable a node to form robust multi-hop networks. More recent developments support multiple radio frontends with vastly varying characteristics (e.g., Bluetooth and a low-power radio) to combine the advantages of these technologies.

Most radios used for sensor nodes operate in license-free ISM bands (e.g., 868 MHz, 915 Mhz, 2.4 GHz). Typical frequencies range from few hundred MHz to few GHz. Various kinds of antennas are used for these frequencies: simple wires, antennas integrated into the circuit board (e.g., "F" or "L" shaped patch antennas), or more compact chip antennas. The size of a simple wire antenna is typically a quarter of the wavelength, (i.e., between 5 and 10 cm for the above frequencies).

Sensor/Actuator Interface

Analog sensors map a certain physical quantity (e.g., temperature) to a variable voltage or current. An analog-to-digital converter (ADC), which is often included in the microcontroller, maps this analog quantity to a digital number. Digital sensors do already include an ADC and do often support a bus system such as I2C. The same applies to actuators (e.g., LEDs, speaker, buzzer): some expect an analog input, other expect digital input and can be connected to standard bus systems.

Most sensor node designs do only provide a minimal set of sensors and actuators (e.g., light sensor, LEDs) on board, mainly for testing purposes. An extension

slot then allows to connect more sensors or even so-called sensor boards. Besides sensors, such extension boards may contain ADC, multiplexers to support many sensors, and even an additional microcontroller to connect analog sensors to standard bus systems.

Real-time Clock

Since microcontrollers are clocked circuits, an additional hardware clock is not strictly necessary. However, some processor cores support dynamic frequency scaling for power efficiency. Also, the microcontroller may be switched to power-saving sleep modes. Therefore, an additional external real-time clock chip and oscillator is often used to implement a stable clock that is independent of a system clock with variable frequency.

Battery and Power Supply

Sensor nodes may be powered by batteries or may scavenge energy from the environment (e.g., vibrations, light). However, the power output of many energy sources often varies over time (e.g., output voltage of a battery, solar cell) and often does not match the requirements of the sensor node electronics. Additional power supply circuitry is needed to transform the output of these power sources. For example, a so-called step-up converter can be used to power 3.3V circuits from a single 1.5V battery. It has also been observed that the effective capacity of a battery can be increased if power is drained in bursts rather than continuously, allowing the battery to recover between bursts. An implementation of this strategy requires conversion circuitry to provide a constant voltage to the sensor node electronics. Many sensor node designs do also provide switchable power supplies for individual subsystems (e.g., radio, sensors) in order to optimize power consumption.

Co-Processors

Microcontrollers may not be sufficient to perform complex signal processing tasks (e.g., FFT, correlations) which are often required for preprocessing high-volume sensory data on the sensor nodes. Hence, some sensor node designs provide FPGA or DSP [73, 130] add-ons for this purpose.

2.5.2 Egrains

Some applications may require the functionality of Motes as described in the previous section, but the size of existing prototype systems makes the use of these devices inconvenient (e.g., body-worn sensors in healthcare applications) or impossible. Hence, there is a need for Egrains – devices that provide the functionality of a Mote within the size of a cubic centimeter and less. Currently, two approaches for achieving the desired size reduction are examined: micro-integration techniques and system-on-chip solutions.

Micro integration seeks to improve the packaging density of discrete circuits [74, 114]. This can be achieved in various ways. The simplest approach is to use more size-efficient chip packaging techniques. One example are so-called Flip Chips (FC), where the leads are on the bottom of the chip and hence do not require extra space as do standard Surface Mounted Devices (SMD). Another technique to construct more compact devices is 3D integration, for example by stacking multiple boards, or by using flexible boards which can be folded into a stack or other 3D layouts. Yet another option are so-called Multichip Modules (MCM), where bare silicon dices (without plastic packaging) are used to construct a circuit. The resulting circuitry is than sealed as a whole with plastic.

In contrast to micro-integration techniques, system-on-chip (SOC) solutions combine the various circuits on a single silicon die, such that only a few external components (e.g., quartz oscillator, antenna, power supply) are required to build a complete sensor node [131, 138]. In the SOC design process it is possible to take a modular approach, where the various circuits (e.g., microcontroller, radio) are just laid out and connected on a single piece of silicon without further integration of the circuits. Otherwise, the development process of SOC is similar to Application-Specific Integrated Circuits (ASIC).

With both approaches, there are two fundamental limitations for the achievable size reduction: antennas and batteries. The size of radio antennas is directly linked to the wavelength of the carrier signal and is a couple of centimeters for commonly used ISM bands. The use of higher frequency bands (e.g., some tens of GHz) is possible, but does increase the power consumption of the radio transceiver due to higher signal attenuation and due to higher dielectrical losses. Since the radio is the dominating power consumer of a sensor node, savings in antenna size will typically result in more battery volume.

Since the above miniaturization methods themselves do not significantly reduce power consumption, power sources with a higher energy density would be required to reduce the size of the power supply. However, the energy density of electro-chemical power sources (e.g., batteries) did only improve by 20 % during the last twenty years and a significant improvement of this rate in the future is not expected.

2.5.3 Smart Dust

Some applications may require even smaller sensor nodes that resemble the size of a dust particle. Some applications that may require such tiny devices are the integration of sensors into coatings (e.g., paint), applications where sensor nodes have to float in air, or applications where the presence of sensor nodes must be hardly noticeable (e.g., military).

As explained in the previous section, the use of radio communication presents some fundamental limits to the integration density of sensor nodes. While Egrains are just shrunk versions of motes, Smart Dust will likely require the use of other technologies. This applies in particular to communication techniques.

A project at UC Berkeley examined the use of laser-based communication for

Smart Dust. A similar approach is also examined by the Speckled Computing initiative [132]. As described in [49], Berkeley Smart Dust nodes consist of a small battery, a solar cell, a power capacitor, sensors, a processing unit, an optical receiver, and a corner-cube retroreflector (CCR) within a space of few cubic millimeters. Later versions might also contain an active transmitter based on a semiconductor laser diode. However, the high power consumption of the laser diode may significantly limit the value of such a component. Therefore, in the near future, communication will be possible only between sensor nodes and a so-called base station transceiver (BST).

The BST mainly consists of a steerable laser and a compact imaging receiver. For downlink communication, the BST points the modulated laser beam at the optical receiver of a node. For uplink communication, the BST points an unmodulated laser beam at the node, which modulates the laser beam and reflects it back to the BST using its CCR. Using its imaging receiver, the BST can receive and decode transmissions from dust nodes.

The CCR is a special Micro Electro-Mechanical Systems (MEMS) structure consisting of three mutually perpendicular mirrors. The CCR has the property that any incident ray of light is reflected back to the source under certain conditions. If one of the mirrors is misaligned, this retroreflection property is spoiled. The Smart Dust CCR includes an electrostatic actuator that can deflect one of the mirrors at kilohertz rates. Using this actuator, the incident laser beam is on-off modulated and reflected back to the BST. Using a 5-milliwatt laser, data transmission at a bit rate of up to 1 kilobit per second over a range of up to 150 meters in full sunlight has been demonstrated [49].

This type of design implies a single-hop network topology, where the nodes cannot directly communicate with each other, but only with the base station. The base station can be placed quite far away from the nodes. Communication may suffer from significant and highly variable delays if the laser beam is not already pointing at a node which is subject to communication with the BST.

Obviously, this communication scheme has a number of limitations. It requires a free line of sight between BST and Smart Dust nodes, and nodes must point their optical receiver and CCR towards the BST. On the other hand, communication is very energy efficient, since the dust nodes do not actively emit any signals. Also, the complexity of the transceiver and hence also its energy consumption is rather low. Moreover, optical receiver and CCR are small enough to fit into few cubic millimeters.

Recent prototypes of Smart Dust [105] implement the optical receiver, CCR, a light sensor, a solar cell, and a simple control unit within 16 cubic millimeters. Future devices are expected to include a complete processing unit instead of the simple control unit, provide a wider variety of sensors, and are expected to feature further reductions in size and energy consumption.

2.5.4 Commodity Devices

In the previous sections, we have described devices that have been deliberately developed for use as sensor nodes. However, many existing sensor network applications do also use existing commodity devices such as mobile phones, PDAs, embedded PCs, or even cameras and laptop computers. One reason for this is that many developers prefer to develop early prototypes of an application using such commodity devices, since software development for these devices is often more convenient due to established operating systems and tool chains.

Commodity devices may also be included in heterogeneous sensor networks for a number of reasons [14, 96]. Firstly, commodity devices may act as a gateway between the sensor network and a background networking infrastructure. A mobile phone, for example, may connect a sensor network to the GSM network. Secondly, many commodity devices offer a user interface that can display monitoring results to the user or which allows a user to analyze, control, and debug certain aspects of a sensor network. Thirdly, many commodity devices offer sensors (e.g., audio, photo, video) and actuators (e.g., audio output, phone ringing) that may be useful in certain applications. And finally, commodity devices typically offer rich computing and storage resources to perform more complex tasks. For example, PDAs and embedded PCs may be used as cluster heads or in the upper layers of tiered networks, where they serve a large number of more constrained sensor nodes.

The use of commodity devices in conjunction with sensor networks requires the provision of communication links between these devices and other sensor nodes. If the sensor network uses standardized communication technology such as Bluetooth, commodity devices with support for this technology can be directly integrated into the network. If the sensor network uses a networking technology which is not directly supported by a commodity device, a sensor node can often be connected to the commodity device via serial IO, such that the attached sensor node acts as an external network interface for the commodity device.

2.6 Technical Challenges

The characteristics of wireless sensor networks can present a number of major challenges to the development of algorithms, protocols, and systems. The existence and impact of these challenges varies across the design space. Below we present the main technical challenges and their relationship to the dimensions of the design space.

2.6.1 Resource and Energy Constraints

Each application has specific constraints on the size and prize of individual sensor nodes. These constrains directly imply constraints on the available amount of energy, computing (instructions per time unit), storage (amount of memory), and communication resources (bandwidth, range) per sensor node. On the other

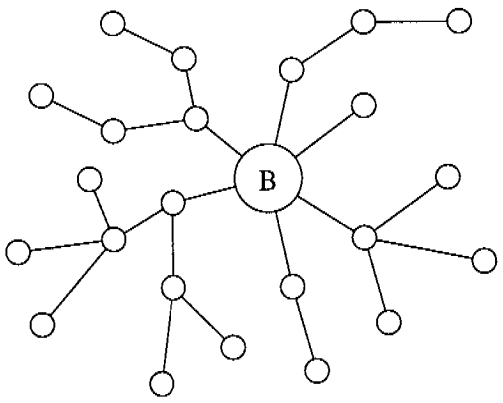


Figure 2.1: Sensor nodes send data to base station B along a spanning tree. Nodes close to the base station will run quickly out of power due to forwarding messages from nodes further away.

hand, a task to be performed by a sensor node has particular requirements on the above resources. For example, a required minimum lifetime maps directly to a requirement on available energy, given a certain power consumption per time unit. Different implementations of the same task can have vastly varying requirements. Therefore, the tighter the constraints on available resources and energy, the more attention must be paid to resource-efficient and energy-efficient implementations. In many cases, resource and energy efficiency become the primary design goals, which renders existing solutions for many problems useless, since they have not been designed for resource and energy efficiency. In particular, many common technologies (e.g., GPS) may not be applicable due to their high cost, size, and energy overheads. As explained in Section 2.5.3, the resource requirements of radio communication may preclude its use in very constrained setups such as Smart Dust.

Often it is also important to ensure that resource usage and energy consumption is equally spread among the nodes of the network. If some nodes exhaust their battery quickly and fail early, resulting permanent network partitions may render the network inoperational. Likewise, hotspot usage of resources may lead to bottlenecks such as network congestions. Figure 2.1 illustrates a typical problem with asymmetric energy consumption in sensor networks. Sensor nodes send sensor readings along a spanning tree to a base station B for evaluation. Nodes close to the base station will run quickly out of power since they forward messages from nodes further away. Failure of these nodes will create a "dead ring" around the base station.

2.6.2 Network Dynamics

Depleted batteries and corruptive environmental conditions (e.g., pressure, humidity, temperature, destructive chemicals) often lead to node failures. Temporary environmental obstructions (e.g., vehicles, humans) may influence the communica-

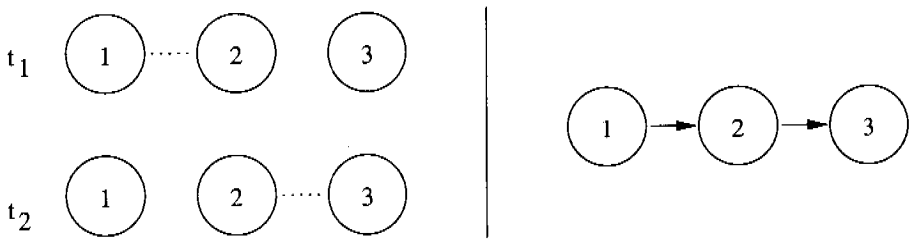


Figure 2.2: Message transport across partition boundaries through node mobility may result in unidirectional multi-hop paths with unpredictable, unbounded delays.

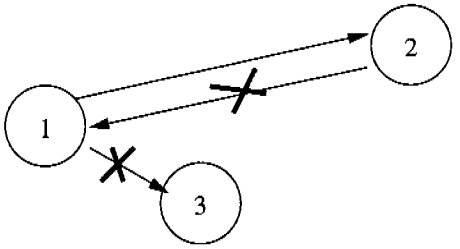


Figure 2.3: Interference may lead to temporal and spatial variations in link quality, possibly resulting in asymmetric and unidirectional single-hop links.

tion range of nodes. Nodes may be mobile, new nodes may be added to replace failed ones. All these issues may lead to frequent topology changes in sensor networks. In particular, temporary network partitions are likely to exists in sparse networks.

Despite intermittent connectivity, messages can be forwarded across partitions by mobile nodes as illustrated in Figure 2.2. At time t_1 nodes 1 and 2 are within communication range of each other only, then node 2 moves towards node 3, such that at time t_2 nodes 2 and 3 are within communication range of each other only. Node 2 can carry a message from node 1 to node 3 across a partition boundary. The resulting multi-hop message flow has two challenging properties. Firstly, the path is unidirectional: it is not possible to send a message from node 3 to node 1 unless there exists a node with an appropriate mobility pattern. Secondly, the delay of this message flow can be arbitrarily high and is hardly predictable unless the mobility pattern of node 2 is known in advance.

It has also been observed that due to interference of sensor nodes among each other and with the environment, communication link quality may vary heavily from node to node, over time and space [39]. This may lead to a number of temporary or permanent effects as depicted in Figure 2.3. Although all nodes are identical, node 2 may not be able send a message to node 1 even if node 1 can send a message to node 2. Also, node 1 may not be able to send a message to node 3, even though node 1 is closer to node 3 than to node 2.

Ensuring robust operation of a sensor network in such setups can be a very challenging task.

2.6.3 Network Size and Density

Depending on the desired observation detail and area, wireless sensor networks may consist of a large number of sensor nodes. This is due to the fact that sensors typically have a small effective range, while at the same time a potentially large geographical area of interest must be covered by the sensors. Also, the quality and detail of observations can profit from redundant deployment of sensor nodes (where many sensors cover a certain physical spot) very close to the phenomenon of interest.

Ensuring that sensor networks scale to large numbers of densely deployed nodes can be technically challenging. If omnidirectional radios are used for communication, for example, the network capacity per node decreases with the node density. In dense networks, the occurrence of a physical event may trigger communication at a large number of collocated nodes, which could lead to network congestions and increased delays.

2.6.4 Unattended and Untethered Operation

Depending on the application, sensor networks may have to be deployed in remote, unexploited, or hostile geographical areas. In such cases, sensor networks may not rely on well-engineered or excessive hardware infrastructure (e.g., for communication, localization, time synchronization).

After deployment, it may be impossible or prohibitively costly to physically access sensor nodes for recharging, maintenance, etc. Hence, sensor networks may have to operate unattended for extended periods of time.

If the sensor network consists of a large number of nodes, manual configuration of individual nodes may not be an option. Additionally, pre-deployment configuration is often infeasible because some configuration parameters such as node location and network neighborhood are typically unknown prior to deployment. Also, node parameters may change over time, necessitating dynamic re-configuration. Hence, a means for in-situ configuration after deployment is often necessary. The term self-configuration is commonly used to express the fact that a sensor network should configure itself without manual intervention.

2.7 Design Principles

In the previous section we outlined a number of technical challenges. Fortunately, there exist general techniques to deal with these challenges or to mitigate their effect.

2.7.1 Adaptive Tradeoffs

Often, there are tradeoffs between different system parameters such as fidelity, latency, and energy/resource usage. Improving one of these parameters often implies

a degradation of other parameters. The requirements on these parameters do not only vary among different applications, but also during the lifetime of a single application. Algorithms that provide "turning knobs" to dynamically adapt these tradeoffs to the actual application requirements can help optimize the resource and energy consumption, for example. If such a parameterization of a single algorithm is limited or impossible, the provision of a family of algorithms – with different, but fixed tradeoffs – to select from may be worthwhile option.

2.7.2 Multi-Modality

The concurrent use of different approaches (i.e., modalities) to the same problem can often help to improve the robustness. For example, wide-band communication techniques use multiple frequencies concurrently or sequentially (e.g., frequency hopping, chirp sequences) to improve the robustness to narrow-band interferences. The detection of an object can be made more robust by using different types of sensors (e.g., a motion detector and a magnetometer).

2.7.3 Local Interaction

Limiting the interaction of each sensor node to nodes in the direct network neighborhood can improve scalability, since the number of nodes in the local neighborhood does not depend on the total number of nodes in the network if the node density is fixed.

2.7.4 Data Centricity

In many traditional distributed systems, algorithms such as routing and distributed data storage operate independently of the actual contents and meaning of the data. In contrast, data-centric approaches make the type of data (e.g., "temperature reading") and its properties (e.g., "sample obtained at position p") an explicit input to such algorithms.

Data centricity can improve the robustness of sensor networks, since it can be used to decouple sensory data from specific sensor nodes. By using a data-centric query such as "what is the current temperature at position p", any node located near position p can answer the query, whereas a traditional address-centric query such as "what is the temperature at node #7" fails if node #7 experiences problems, even though a nearby node with a different address could answer the query.

Data centricity can also reduce energy and resource consumption, since it supports content-based filtering (e.g., "not interested in temperature readings below T degrees") and placement of sensory data (e.g., storage close to a potential user of a certain type of data).

2.7.5 In-Network Data Processing

Instead of sending streams of raw sensory data to a central processor for evaluation, raw sensory data can be preprocessed by sensor nodes (i.e., in the network), as the data flows through the sensor network. For example, a sensor node can filter sensor readings, such that only application-relevant data is sent through the network. Another example is in-network aggregation and fusion of sensor readings that originate from different sensor nodes.

In-network data processing can improve the scalability and can reduce the energy/resource consumption, since it can significantly reduce the data volume that has to be routed through the network.

2.7.6 Cross-Layer Interaction

Traditional networked systems are typically based on layered designs and implementations (e.g., hardware layer, network layer, application layer), where each layer provides a service to the layer on top of it. The layers are mostly independent of each other in order to enable the exchange of single layers without touching the other layers. While such an approach ensures genericity and modularity, it may cause significant overheads, since any single layer has to satisfy the *maximum* of the requirements of all possible instances of the layer on top of it. For example, if the application layer can tolerate the loss of a specific subclass of messages, the network layer cannot do better that retransmitting all messages, since it does not know which messages are important to the application layer and which are not.

Cross-layer interaction relaxes the strict separation of layers by passing useful bits of information from one layer to another. In the above example, the application layer may want to pass a classifier for important messages to the network layer. Hence, cross-layer interaction can improve the energy and resource efficiency.

The information passed across layers can be static (e.g., certain fixed features of the application or hardware) or dynamic. In the static case, a generic approach with large overhead can often be collapsed into a specific and more efficient solution (e.g., application-specific or hardware-specific designs).

The application and hardware layers are particularly valuable sources of information that can improve the efficiency of other layers. For example, the application layer may pass filter criteria and aggregation rules to the network layer in order to enable in-network data processing. Some algorithms in the application layer can exploit knowledge about the quality of a link (e.g., signal strength).

2.8 Summary

This chapter discussed wireless sensor networks in general. We first characterized WSN, also in the context of related research areas such as ubiquitous computing, embedded systems, mobile networking, and digital signal processing. Motivated by a study of concrete applications, we proposed a design space for wireless sensor

networks as a replacement of the currently adopted narrow view. We showed that concrete applications occupy different points in this design space. We discussed implications of the design space and presented four classes of hardware that supports different regions in the design space. We also discussed technical challenges associated with different regions of the design space and sketched design principles that can be helpful in dealing with these challenges.

Chapter 3

Space and Time in Sensor Networks

Space and time play a crucial role in wireless sensor networks, since sensor nodes are used to collaboratively monitor physical phenomena and their spatio-temporal properties. Consequently, a number of techniques and distributed algorithms for location estimation and time synchronization have been developed specifically for sensor networks. However, research in these two domains has been performed by mostly separated research communities.

A closer look on both research domains reveals that there are many similarities. This does affect a variety of aspects of location estimation and time synchronization, ranging from applications and requirements to basic approaches and concrete algorithmic techniques. The purpose of this chapter is to make this close affinity explicit in order to further a better understanding of both domains. We will base our detailed discussion of synchronization and localization in the subsequent two chapters on the common framework we develop in this chapter.

The remainder of this chapter is structured as follows. In Section 3.1 we describe uses of space and time in sensor networks. Section 3.2 presents a common model for location estimation and time synchronization and discusses various requirements and different approaches to location estimation and time synchronization based on this model. In Section 3.3 we examine the structure of distributed algorithms for location estimation and time synchronization. In particular, we will point out that algorithms for both problems are based on a number of common structural elements. Section 3.4 discusses various limitations and trade-offs of this class of algorithms.

3.1 Uses of Space and Time

Figure 3.1 illustrates three important application classes of space and time in sensor networks. Typically, a sensor network is tasked by and reports results to an external observer (a). A sensor network also interacts with the physical world through distributed sensors and possibly also through actuators (c). Finally, the sensor

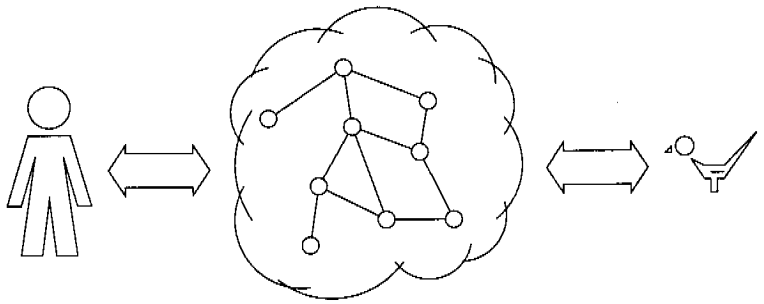


Figure 3.1: Applications of space and time. (a) interaction of an external observer with the sensor network, (b) interaction among sensor nodes, (c) interaction of the sensor network with the monitored real world.

nodes interact among each other to coordinate distributed computations (b). The following paragraphs will discuss applications of space and time in these three domains.

3.1.1 Sensor Network – Observer

In many applications, a sensor network interfaces to an external observer for tasking, reporting results, and management. This external observer may be a human operator (as depicted in Figure 3.1) or a computer system. Tasking a sensor network often involves the specification of regions of interest in space and time such as "only during the night" or "the area south of ...". Since the observer is typically interested in a physical phenomenon of the real world (and not in individual sensor nodes), such spacetime addressing is often preferable over addressing individual nodes or groups of nodes by identifiers.

As a sensor network reports monitoring results to the observer, many spatio-temporal properties of observed physical phenomena are of interest. For example, time and location of occurrence of a reported physical event are often crucial to associate event reports with the originating physical events. Properties such as size, shape, speed, trajectory, density, frequency do all refer to the categories time and space.

3.1.2 Sensor Network – Real World

In sensor networks, many different sensor nodes distributed over an area may be involved in the observation of a single physical phenomenon. One of the key functions of a sensor network is therefore the assembly of many distributed observations into a coherent estimate of the original physical phenomenon – a process known as data fusion. Space and time are key ingredients for data fusion. For example, many sensors can only detect the proximity of an observed object. Higher-level information, such as speed, size, or shape of an object can then only be obtained by correlating

data from multiple sensor nodes whose locations are known. The velocity of a mobile object, for example, can be estimated by the ratio of the spatial and temporal distances between two consecutive object sightings by different sensor nodes. As another example, the size and shape of a widespread object can be approximated by the union of the coverage areas of the sensor nodes that concurrently detect the object.

Since many different instances of a physical phenomenon can occur in spatio-temporal proximity, one of the tasks of a sensor network is the separation of sensor samples, that is, the partitioning of sensor samples into groups that each represent a single physical phenomenon. Spatio-temporal relationships (e.g., distance) among sensor samples are a key element for separation.

Spatio-temporal coordination among sensor nodes may also be necessary to ensure correctness and consistency of distributed measurements [40]. For example, if the sampling rate of sensors is low compared to the temporal frequency of an observed phenomenon, it may be necessary to ensure that sensor readout occurs concurrently at all sensor nodes in order to avoid false observation results. This is also an issue for sensor calibration as explained in [19]. Likewise, the spatial distribution of sensors has an impact on the correctness of observation results. For example, in order to estimate the average of a certain physical quantity over a certain physical area (e.g., average room temperature), it is typically not sufficient to simply calculate the average over all sensor nodes covering the area, because then areas with higher node density would be overrepresented in the resulting average value.

It is anticipated that in the future large-scale and complex actuation functions will be realized by coordinated use of many simple distributed actuator nodes that are part of a sensor network. Similar to distributed measurements, spatio-temporal coordination will then also be an important ingredient for consistent distributed actuation.

3.1.3 Within a Sensor Network

Time and location are also valuable concepts for intra-network coordination among different sensor nodes. As sensor networks are essentially distributed systems, many traditional uses of the concepts of time and location do also apply to wireless sensor networks. Liskov [57] points out a number of uses of time in distributed systems in general such as for concurrency control (e.g., atomicity, mutual exclusion), security (e.g., authentication), data consistency (e.g., cache consistency, consistency of replicated data), and communication protocols (e.g., at-most-once message delivery).

One particularly important example for concurrency control is the use of time-division multiplexing in wireless communication, where multiple shared access to a common communication medium may be realized by assigning time slots with exclusive access to the communicating peers. This may require the participating sensor nodes to share a common view on physical time. A prominent use of spatial

information for network coordination is geographic node addressing and routing, where geographic locations replace the use of node identifiers.

A number of approaches intend to improve energy efficiency by selectively switching sensor nodes or components thereof into power-saving sleep modes. In order to ensure seamless operation of the sensor network despite of this, spatio-temporal coordination among sensor nodes may be required. The algorithm presented in [110], for example, extends the lifetime of dense networks by switching off nodes such that the remaining nodes are sufficient to cover the area of interest. To ensure coverage, node locations must be known. Another way of extending network lifetime is to periodically switch off radio transceivers of sensor nodes, since their power consumption is rather high even when only listening to the network. Temporal coordination is required to ensure that activity periods of sensor nodes overlap in time in order to enable communication (see, e.g., [111]).

Another service of importance for sensor network applications is temporal message ordering [81]. Many data-fusion algorithms have to process sensor readings ordered by the time of occurrence (e.g., in the approach for velocity estimation sketched above). However, highly variable message delays in sensor networks (cf. Section 2.6.2) imply that messages from distributed sensor nodes typically do not arrive at a receiver in the order they have been sent. Reordering messages according to the time of sensor readout requires temporal coordination among sensor nodes.

The close relationship between time and space in the physical world is also reflected by methods for time synchronization and location estimation themselves. For example, methods for location estimation based on the measurement of time of flight or time difference of arrival of certain signals typically require synchronized time. The other way round, location information may also help to achieve time synchronization. This is due to the fact that time synchronization approaches often have to estimate message delays. One component of the message delay is the time of flight of the carrier signal between two nodes, which can be calculated if the distance between sender and receiver and the propagation speed of the carrier signal are known (cf. Section 5.1.1).

3.2 Locating Nodes in Spacetime

In this section we present a common model for location estimation and time synchronization. Using this model, we will discuss various requirements on and different classes of time synchronization and localization.

One possible way to model physical space is to do this as a three-dimensional real-valued vector space. Likewise, physical time can be modeled as a one-dimensional real-valued vector space. These two vector spaces are often combined to form a four-dimensional vector space known as *spacetime*. To indicate points in spacetime, a coordinate system is used, consisting of the vector o (the origin) and four linearly independent vectors e_1, e_2, e_3, e_4 (the axes). To avoid relativistic effects and to simplify our discussion, we assume that a coordinate system has the following properties: $e_4 = (0, 0, 0, t)$, the e_i are mutually orthogonal (i.e., inner

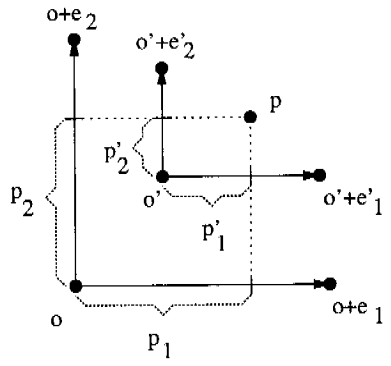


Figure 3.2: A point p in spacetime and its coordinates p_i and p'_i in two different coordinate systems.

product is zero), and $|e_1| = |e_2| = |e_3|$. In other words, the space axes e_1, e_2, e_3 form a Cartesian coordinate system, e_4 is the time axis, and $|e_1| = |e_2| = |e_3|$ and $|e_4|$ are the space and time units, respectively. Any point p in spacetime can now be specified by its coordinates (p_1, p_2, p_3, p_4) with respect to the coordinate system (o, e_1, e_2, e_3, e_4), such that p is given by $o + p_1e_1 + p_2e_2 + p_3e_2 + p_4e_4$.

Under these assumptions, the spatial distance between two points p and q is given by $\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}$, and the temporal distance is given by $|p_4 - q_4|$.

The above model allows a unified view on localization and time synchronization as follows. If a sensor node is modeled as a point p in spacetime, localization and time synchronization can be considered as determining the current coordinates of p with respect to a given coordinate system. We refer to this process as *locating a sensor node in spacetime.*

Note that it is quite common to use different coordinate systems, even in a single application. However, using a simple coordinate transformation scheme, the coordinates p_i of a given point p can be transformed into coordinates p'_i in a primed coordinate system, as depicted in Figure 3.2 for a two-dimensional coordinate system.

In the remainder of the chapter we will simply use the terms *localization / location* as an abbreviation for localization / location in spacetime. We will use *localization / location in time* and *localization / location in space* when specifically referring to time and space.

Localization in spacetime comes in many different flavors and with many different requirements and practical constraints, which are discussed in the following sections.

3.2.1 Internal vs. External

With external localization in spacetime, a given coordinate system is used as a reference. With internal localization, there exists no predefined coordinate system. The nodes of a sensor network then have to agree on a single coordinate system,

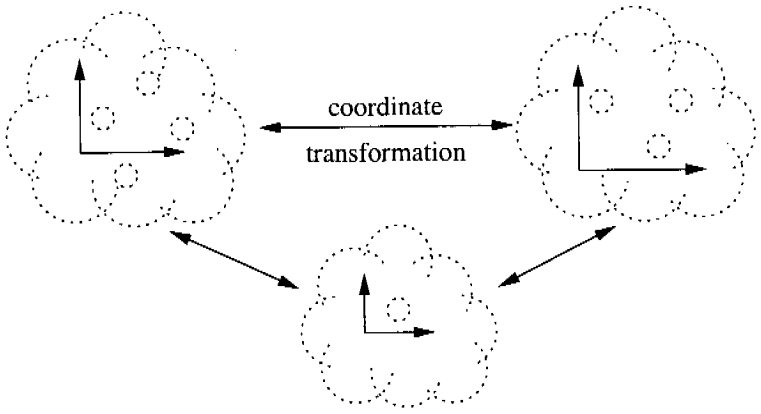


Figure 3.3: Small sets of nodes may use local coordinate systems. If a point in spacetime is passed beyond the scope of such a local coordinate system, a coordinate transformation is applied.

but which one is actually chosen is irrelevant.

Note that external localization is a special case of internal synchronization, since a coordinate transformation can be applied to map coordinates w.r.t. an arbitrary coordinate system used for internal synchronization to coordinates w.r.t. a predefined external coordinate system.

External localization is mostly used when interfacing to the real world and observers, since there are well-established coordinate systems used in daily life such as the coordinate system defined by Universal Transverse Mercator (UTM) space coordinates and Coordinated Universal Time (UTC). For spatio-temporal coordination among sensor nodes, internal localization is often sufficient.

3.2.2 Global vs. Local

In order to be able to compare two points p and q in spacetime, the coordinates of the two points must be known w.r.t. a single coordinate system. The most obvious way to achieve this is to have all network nodes use a single global coordinate system. In this case, any sensor node can easily compare any two points in spacetime obtained from any two nodes.

However, the use of a single global coordinate system is not the only possible solution. As illustrated in Figure 3.3, small sets of nodes or even single nodes may use a local coordinate system each. If points in spacetime remain within the scope of such a local coordinate system, they can be easily compared. However, if the coordinates of a point in spacetime are passed across the border between the scopes of two different local coordinate systems, a coordinate transformation must be applied to the point.

When to prefer the one or the other approach depends on the actual application. Maintaining a coordinate system across a set of distributed network nodes requires communication among the participating nodes. However, comparing points within the scope of a single coordinate system then comes for free. Local coordinate

systems typically do not require active communication among nodes using different coordinate systems. Passing points across a coordinate system boundary, however, requires to compute a suitable transformation between the two involved coordinate systems, which then has to be applied to the affected points.

Therefore, a reasonable approach would be to cluster nodes based on their interaction patterns, where each cluster has a local coordinate system. If two nodes do frequently exchange points in spacetime, they should end up in the same cluster. If there is strong interaction among all nodes in the network, using a single global coordinate system is likely the better choice.

3.2.3 Point Estimates vs. Bounds

Actual implementations of localization in spacetime are based on measurements. Since measurements are always afflicted with errors, only estimates of the coordinates of a point in spacetime can be obtained in practice. Despite this, it is often convenient to use such point estimates as if they were correct in the absolute sense.

Another approach is to explicitly deal with errors in measurements by specifying bounds on the actual coordinates of a point in spacetime, where one assumes that the true value lies within the bounds. Common ways of specifying bounds are bounding boxes and spheroids. In the special case of (one-dimensional) time, both map to intervals.

Both point estimates and bounds have advantages and disadvantages that influence the choice of one over of the other. Basically, point estimates are convenient to use due to the simplicity of point arithmetic and because statements in terms of the abstract spacetime model can be directly applied to the point estimates. However, the use of point estimates may lead to wrong results. For example, for two point estimates $\hat{p}$ and $\hat{q}$ where $\hat{p}_4 < \hat{q}_4$ holds, p may despite of this actually represent a later point in time than q.

While the use of explicit bounds is often more complex and inconvenient, and sometimes rather imprecise, errors like the above one can be avoided. However, the use of bounds also introduces situations where it is impossible to decide on a certain predicate. For example, if intervals are used to represent points in time, it cannot be determined whether one point is earlier than another if the corresponding intervals overlap. While the introduction of such undecidable situations may seem undesirable from a technical point of view, they explicitly represent fundamental limitations of the system and alert the application or user about it, instead of making arbitrary and potentially wrong decisions.

Yet another approach to deal with the imprecision of localization algorithms is the use of probability distributions over spacetime. However, due to the practical difficulty of dealing with probability distributions, this approach is currently barely used in distributed algorithms for sensor networks.

3.2.4 Points vs. Distances

In the previous discussions we assumed that points in spacetime originating from different nodes in the sensor network need to be compared. However, there are also applications where individual nodes locally measure distances between points in spacetime and where it is sufficient to compare distances measured at different sensor nodes.

As an example, consider an application where sensor nodes measure the time during which a certain phenomenon can be sighted and where the sighting durations at different sensor nodes must be compared (e.g., to estimate the acceleration of a mobile object). Here, the actual points in time when the phenomenon appeared or disappeared are irrelevant. However, it is important that different sensor nodes measure the same duration given identical physical stimuli.

Obviously, measuring and comparing distances is a special case of measuring and comparing points in spacetime, since a distance can be easily calculated when the points are given w.r.t. a common coordinate system.

3.2.5 Scope and Lifetime

A *scope* defines a subset of nodes where localization in spacetime is required. A *lifetime* defines a time interval during which localization is required. The two extremes are everywhere/continuous and on-demand, where localization is only performed where and when actually needed.

Both lifetime and scope requirements can vary from application to application and may change dynamically and in unpredictable ways. In many sensor network applications, scope and lifetime are correlated with the occurrence of the observed physical phenomena. For example, to locate an object moving through a sensor network, nodes that detect the object might define the scope and the lifetime.

With everywhere/continuous localization, the localization procedure is performed permanently on all nodes, such that an up-to-date estimate of the current location in spacetime is immediately available whenever requested by the application. With on-demand localization, the localization procedure is performed only where (i.e., on a certain node) and when the application requests the current location in spacetime. The result is only available after a delay caused by the execution of the localization algorithm.

The overheads of the two approaches depend on the frequency of the application requesting localization. If rarely requested, on-demand localization may be more efficient. If frequently requested, continuous localization is likely to be more efficient. In sensor networks, where activity is often triggered by the occurrence of rare physical events, the on-demand approach is certainly a promising technique for achieving resource efficiency.

3.2.6 Precision

A localization algorithm yields a point estimate or bounds on a sensor node's actual position p in spacetime. Precision is a measure for how well this result matches the ground truth locations p of nodes across the network over time. For algorithms returning point estimates, the *instantaneous* precision for a given node at a given point in time is usually expressed in terms of the distance between the point estimate and p. Algorithms that return bounds are error-free if p is actually enclosed by the bounds. However, the precision of bound-based algorithms can be expressed by the uncertainty of the bounds (e.g., the volume of a bounding box, the length of an interval).

To derive the overall precision of an algorithm within a given scope and during a given lifetime, the instantaneous precisions of the nodes have to be combined. The combined precision then has to be accumulated over time to arrive at a single value that characterizes precision. Common ways of combining instantaneous precision values of many nodes are maximum, average, and standard deviation. A variant often found in the literature is the maximum error after removing a given percentage (e.g., 5%) of the largest errors. The combined precision typically improves during the execution of an algorithm and approaches a stable value in the steady state. The combined precision in the steady state can be used to express the overall precision of an algorithm.

Requirements on the precision may heavily vary from application to application. This applies both to quality and quantity. With respect to quality, an application might require a certain average precision, other applications may request a certain maximum error. The requirements on the distribution of precision over the network and over time may also vary from application to application. With respect to quantity, required precision is closely related to the temporal frequency and spatial detail of the phenomena that require localization in spacetime. For localization in time, precision requirements range from a maximum error of few micro seconds (e.g., for controlling access to the communication channel) to seconds or even minutes (e.g., for activating a sensor network during certain times of the day). With respect to location, precision requirements range from a maximum error of some centimeters (e.g., locating a shooter [97]) to tens or even hundreds of meters (e.g., locating an animal herd).

As mentioned in Section 3.2.5, the scope of localization in sensor networks is often defined by a set of collocated sensor nodes that cooperate in monitoring a close-by event in the real world. For this kind of application, the precision among this set of collocated nodes typically must be high. However, the precision among nodes which are far apart in space may be of lesser importance. We will return to this issue in Section 3.4.1.

3.2.7 Other Quality-of-Service Aspects

Besides the aspects discussed so far, a number of additional QoS characteristics of localization in spacetime are of practical relevance. Two prominent examples are

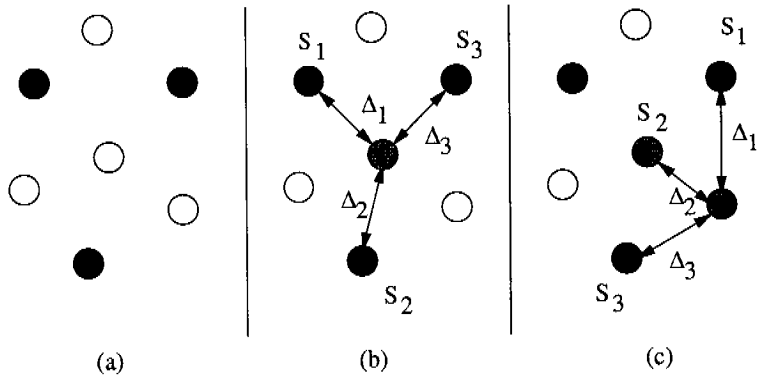


Figure 3.4: Client nodes infer their location in spacetime by measuring spatio-temporal relationships Δ (e.g., Euclidean distance, message delay) to black reference nodes with known locations S in spacetime. The process is iteratively applied. The figure shows from the left to the right, a sequence of three snapshots.

robustness and security. A robust localization algorithm delivers correct location estimates even in the presence of well-defined, accidental failures. Another aspect is secure verification of location estimates, where spoofed locations can be detected (see, e.g., [21]).

3.3 Distributed Algorithms for Localization in Spacetime

Many practical distributed algorithms for localization in space (e.g., [17, 36, 69, 75, 88, 89]) and time (e.g., [26, 38, 60, 98, 101]) are based on a few common structural elements. In this section we point out these structural elements and discuss various concrete instances of these elements found in existing algorithms.

Consider the example illustrated in Figure 3.4. Part (a) shows two kinds of nodes: black *reference nodes* with known locations and white *client nodes* with unknown locations. In part (b), a gray client node measures its distance Δ_i from a number of neighboring reference nodes. Using the locations S_i of the references and the measured distances Δ_i, the gray node infers its own location in spacetime. The client node can now also act as a reference for other client nodes in subsequent iterations of the algorithm as illustrated in part (c). Eventually, all nodes should be able to measure distances to a sufficient number of neighboring reference nodes in order to estimate their location in spacetime.

The meaning of the symbols Δ and S has to be interpreted in a rather broad sense here. S is any state information of a node that is relevant to a localization algorithm. Examples for S are time, location, orientation, and address of a node. S may also include confidence values that characterize the precision of the respective bits of state information. Δ is a spatio-temporal relationship between a client node and one or more reference nodes. Examples include Euclidean distance, hop distance, message delay, and angle with respect to the orientation of the client. Δ

may also include confidence values.

A pair (S, Δ) can be interpreted as a *constraint* on the possible spacetime locations of a client node. For example, if S is a location of a reference node in space and Δ its Euclidean distance, then the location of the client node is constrained to the hull of a sphere with radius Δ centered at S. As we will show in Section 3.3.2, a constraint may also involve multiple reference nodes, such that Δ is a relationship among a client node and any number of reference nodes (e.g., client node is closer to reference 1 than to reference 2). Also, reference nodes need not be network neighbors of the client node.

A second structural element of localization algorithms is a procedure for *combining multiple constraints*. As pointed out above, a single constraint limits the possible locations of a client node, but the resulting solution space often does not satisfy precision requirements. Hence, multiple constraints have to be combined (e.g., intersected) to further cut down the solution space (e.g., to a single point in spacetime).

A third important component of localization algorithms are rules to *select constraints*. In dense networks with many reference nodes, there is a large set of possibilities for obtaining constraints that involve different sets of reference nodes. While a large number of constraints may result in very precise location estimates, the overhead for combining such numerous constraints may be prohibitive. Hence, the goal is to select a small number of tight constraints that are sufficient to achieve a certain precision. This selection process is not trivial, as it depends on a number of parameters such as the precision of the state information of the individual reference nodes, but also on a particular combination of reference nodes. Also, certain reference nodes may only become available after they have estimated their location themselves. Often, an overlay structure (e.g., spanning tree, clustering) is constructed to ease this selection process. For example, a client node may use its parent in a spanning tree as a reference node. Essentially, constraint selection can be interpreted as the approach an algorithm takes to structure localization in multi-hop networks (i.e., across space).

The fourth important element of localization algorithms is an approach to *maintain localization over time*, since a single estimate of a node's location in spacetime is quickly invalidated due to the progress of time and due to node mobility. The conceptually simplest approach to this problem is to repeat a one-shot localization frequently.

Last but not least, a *bootstrapping mechanism* is needed to provide initial reference nodes that act as seeds for distributed localization algorithms.

An algorithm for localization in spacetime can often be considered as a combination of concrete instances of the above five categories and additional "glue" elements. Many practical algorithms consist of several phases in order to improve precision or other performance metrics. In each phase, different instances of the five categories may be used. For example, several algorithms consist of a first phase to obtain rough location estimates for all nodes. In a second phase, the so-called refinement phase, these initial estimates are further improved.

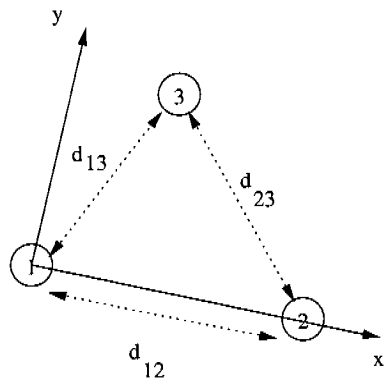


Figure 3.5: Three non-collinear nodes with known mutual distances d_{ij} define a coordinate system for two-dimensional space.

In the following sections we give a more detailed overview of the above structural elements. Sections 4.1 and 5.1 discuss concrete instances of these elements for localization in time and space, respectively.

3.3.1 Bootstrapping

Obtaining constraints typically requires a number of reference nodes with known locations in spacetime. Bootstrapping consists in providing such initial reference nodes with location estimates. The most commonly used approach to solve the bootstrapping problem is the provision of so-called *anchor nodes* which are able to estimate their locations by means of an out-of-band localization mechanism such as GPS, which can provide locations in both time and space. While anchors are a natural way to solve the bootstrapping problem and allow for good precision due to providing location "fixpoints" throughout large networks, they also come with a significant overhead: a certain portion of the nodes must be equipped with additional hardware (e.g., GPS receivers) and an additional infrastructure is often needed (e.g., GPS satellites). We will discuss issues with anchors in more detail in Section 3.4.1.

It is also possible to solve the bootstrapping problem without the use of anchors. Consider for example Figure 3.5, where three nodes 1, 2, and 3 with mutual Euclidean distances d_{12}, d_{23}, d_{13} are depicted. The nodes define a coordinate system as follows. The origin is given by the position of node 1. The positive x axis is given by a ray starting at node node 1 passing through node 2. The positive y axis is given by a ray starting at node 1 that is orthogonal to the x axis and that extends into the half plane (defined by the x axis) that contains node 3. In this coordinate system, the coordinates of the three nodes are $(0,0)$, $(0,d_{12})$, and $(X, \sqrt{d_{12}^2 - X^2})$, respectively, with $X = (d_{12}^2 + d_{13}^2 - d_{23}^2)/2d_{12}$ (e.g., [20, 79]).

Note, however, that this is only one possible coordinate system, any other coordinate system could have been used as well. Hence, in contrast to anchor-based approaches, anchor-free approaches are not suitable for external localization (cf.

Section 3.2.1). Also, the above coordinate system changes when one of the initial reference nodes moves, invalidating the location estimates of all nodes whose positions have been estimated with respect to this coordinate system. The precision of anchor-based algorithms is often superior to anchor-free approaches, since anchor nodes may be distributed over the network to act as fixpoints for localization. With anchor-free approaches, nodes far away from the reference nodes that define an initial coordinate system may experience significant imprecision due to accumulating errors.

3.3.2 Obtaining Constraints

The general form of a constraint is $(\{S_1, ..., S_N\}, \Delta)$, where N reference nodes and their respective state information S_i are involved. Δ represents a spatio-temporal relationship among these reference nodes and the client node. While S_i are typically retrieved from a reference node by means of message exchanges, Δ is usually a measured quantity. In most cases, Δ is either represented by a point estimate (e.g., distance $= X$) or by bounds (e.g., distance $> X$ and/or distance $< Y$).

For localization in time, the delay of network messages is typically used as a foundation for Δ measurements. For example, a round-trip message exchange between client and reference can be used to derive lower and upper bounds on the message delay between reference and client (e.g., [80]). The average of these bounds can be used as a point estimate of the temporal distance (e.g., [38, 101]).

For localization in space, distance-dependent properties of propagating signals (e.g., sound, radio) such as received signal strength or time of flight are typically used as a foundation for Δ measurements. Two common forms of constraints are based on Euclidean distances (e.g., bounds or point estimates for the distance from a reference) and angles (e.g., bounds or point estimates for the direction of arrival of a signal from a reference). Two common constraints that involve multiple references are "closer to" relationships (e.g., client is closer to reference 1 than to reference 2) and distance differences (e.g., client is X meters closer to reference 1 than to reference 2).

3.3.3 Combining Constraints

A single constraint can be interpreted as a region in spacetime that contains the location of a client. Combining multiple constraints typically consists of two steps. In a first step, "bad" constraints are eliminated from the set of available constraints. One example of such bad constraints are outliers that represent a region in spacetime that does not overlap with the regions defined by some or all other constraints. After this step, the remaining constraints should have a non-empty intersection that contains the prospective location estimate of the client.

In a second step, the intersection or a point in the intersection of the remaining constraints is computed. In many cases, this can be achieved analytically, for example by solving an equation system. In some cases, a closed-form solution

cannot be derived or the computational overhead may be prohibitively high. An approximative solution that trades off computational overhead for memory is to subdivide the solution space into pixels, where the intersection region is defined by the pixels that are contained in all constraints (e.g., [44]).

Due to measurement errors, it may happen that there is no sufficiently large subset of constraints with a non-empty intersection. This is often the case if the constraints use Δ relationships that are point estimates (e.g., distance $= X$) rather than bounds. Here, an optimization problem may be set up that requires the solution point to minimize a certain error metric. A commonly used error metric is the distance between a point and a constraint, which is defined as the minimal distance to any point contained in the region defined by the constraint. A typical objective function for the optimization problem is then to minimize the sum of the squared distances between the solution point and each constraint.

For localization in time, constraints define regions that are either points or intervals. In case of intervals, the intersection interval can be computed as the maximum of the lower bounds and the minimum of the upper bounds. In case of point estimates, the average of all constraints is typically used, which minimizes the sum of the squared error distances.

Let us consider some commonly used examples for constraint combination in algorithms for localization in space. A very simple approach is based on centroids (e.g., [17]), where multiple distance-bound constraints are given (i.e., distance from reference is at most X). Here, each constraint defines a sphere. A point close to the intersection of such a set of spheres can be obtained by computing the centroid of the locations of the according reference points.

Another commonly used approach is multilateration to combine multiple distance constraints, where the region defined by each such constraint can be interpreted as the hull of a sphere. Multilateration finds the intersection point of a set of at least 4 spheres in three-dimensional space. In case of exactly four spheres, a linear equation system can be derived and solved to find the intersection point. For more than 4 constraints a minimum-square-error optimization problem can be derived, also resulting in a linear equation system (e.g., [71, 88, 89]).

One further approach is based on triangle tests, where a node performs a check to see whether it is located inside the triangle formed by three reference nodes (e.g., [44]).

3.3.4 Selecting Constraints

At each point during the execution of a localization algorithm, a certain set of reference nodes are available. Using these reference nodes, a number of "good" constraints must be selected out of the large set of possible constraints. This selection is based on the quality of the state information of the reference nodes, on the quality of the spatio-temporal relationship and on temporal aspects. For example, a better set of references may become available in a future iteration. However, the algorithm might not be able to proceed if a node chooses to wait for

better references to become available, since the node itself then cannot act as a reference for other nodes.

There are two different ways to approach this problem. *Structured* approaches first construct an overlay topology that controls selection of reference nodes and triggers client nodes to start measurements. A common overlay topology are trees. For each anchor, a spanning tree of the network is constructed with the anchor at the root. A client node becomes active as soon as its parent has estimated its location and can thus act as a reference (e.g., [26, 38, 101]). Another typical overlay topology are clusters, where nodes in a cluster establish a local coordinate system and estimate their locations in terms of this reference grid. Adjacent clusters must share a number of nodes to allow for the derivation of a coordinate transformation between these clusters (e.g., [20, 33]).

While such structured approaches guide the selection of reference nodes, there is an additional overhead for constructing and maintaining the overlay topology. For example, if nodes fail or move, the overlay topology has to be updated to reflect this change. In contrast, *unstructured* approaches do not explicitly construct an overlay topology (e.g., [60, 89]). Instead, each node actively monitors its neighborhood for a sufficient set of references to become available. While this approach avoids the overheads of topology construction, it introduces an overhead due to a potentially large number of constraints.

Approaches for localization in time often use structured approaches, since a small number of constraints is usually sufficient to achieve the requested level of precision. With localization in space, significant measurement errors and a high degree of freedom due to the three dimensions of space typically requires the use of as many constraints as possible. Hence, many approaches for localization in space are unstructured.

3.3.5 Maintaining Localization over Time

A single run of a localization algorithm allows each node to estimate its location in spacetime at a certain point in real time. However, as time progresses, the precision of this one-shot estimate may decrease quickly due to node mobility or due to the progress of time. Obviously, an algorithm can be executed one more time to obtain up-to-date estimates. The resulting precision over time then depends on the frequency of execution. However, since each execution of the algorithm takes a certain amount of time, this frequency cannot be arbitrarily increased. Hence, the maximum precision over time is also limited. Alternatively, if a certain target precision is requested by the application, the execution frequency may be calculated to be just high enough to provide the requested precision (see, e.g., [101]). For localization in space it is also possible to limit re-execution to nodes that have changed their location (e.g., [88]) in the meantime.

One way to further improve precision over time is the use of sensors to measure the location in spacetime locally without referring to other nodes. This technique is also known as *dead reckoning*. Hardware clocks, for example, are dead-reckoning

devices for estimating the current time. Accelerometers may be used to measure movements and can hence provide estimates of the current position in space (see, e.g., [102]). However, dead-reckoning techniques typically suffer from significant errors that accumulate over time and can therefore only be used to bridge the short gap between two consecutive runs of a localization algorithm. For example, typical hardware clocks suffer from an unknown clock drift between 10 and 100 parts per million. After one minute, the deviation from real time is then between 0.6 and 6 milliseconds. For location estimation using accelerometers, there is a quadratic relationship between acceleration-measurement errors and errors in the computed location estimate.

Another way of improving the precision is *prediction*, where based on location estimates from the past a current estimate is computed. Besides the past behavior, prediction requires a model of how a node can move through spacetime. With respect to time, such a model is rather simple as real-time progresses at a constant rate. The situation gets more complicated for space, where nodes can move in complex patterns. However, it is often possible to derive constraints on the possible locations (e.g., only on roads), bounds on speed and acceleration. For example, if there is an upper bound on the speed of a node, we can derive bounds on the possible locations of a node at time t_1 given the node's location at time $t_0 < t_1$. Technically, prediction can be achieved by fitting a curve (often a polynomial with low degree) to a set of locations in spacetime observed in the recent past. As with dead reckoning techniques, prediction often experiences significant errors.

3.4 Limitations and Trade-offs

Sensor networks are subject to various challenges that have to be met by algorithms for localization in spacetime. In the following subsections we discuss typical trade-offs and limitations of distributed algorithms for localization in spacetime with respect to the technical challenges presented in Section 2.6.

3.4.1 Anchor Infrastructure

In many applications, sensor networks have to be deployed in remote, unexploited, or hostile regions. Sensor networks therefore often cannot rely on sophisticated hardware infrastructure. However, anchor-based algorithms require an anchor infrastructure, where the number, distribution, and arrangement of anchor nodes in a network is a key parameter for the algorithm performance. In this section, we discuss various issues with such an anchor infrastructure.

In order to obtain precise location estimates, the anchors must define an unambiguous coordinate system at the least. With respect to time, the local time scale of any single node defines such an unambiguous time coordinate system. For space, at least four anchors are required. Three anchors typically result in two possible coordinate systems, but one of them can often be excluded due global constraints on possible locations of nodes.

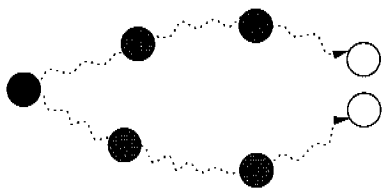


Figure 3.6: Collocated nodes (white) may end up with a large relative error due to using different chains of reference nodes (gray).

However, such a minimum number of anchors is often not sufficient. Energy considerations and interference issues often limit the effective range of anchors. With radio communication, for example, the energy consumption grows with range to the power of k, where typically $2 \leq k \leq 4$. Hence, in large networks with small anchor range, typically a significant portion of nodes cannot directly obtain constraints for a sufficient number of anchor nodes. In this case, an iterative approach can be applied, where nodes first estimate their locations using anchors and then act as "secondary" references for other nodes. As measurement errors accumulate along such chains, the error in the estimated location is the larger, the more iterations are required (i.e., the larger the distance to the anchor is). Depending on the precision of the Δ, this error can be significant. With some approaches for measuring Δ in practice (e.g., measuring Euclidean distance based on received radio signal strength), the error can be as high as 50% of the true distance in realistic settings [88].

One particular problem with using a small number of anchors is that collocated nodes may end up with large relative errors due to using different chains of reference nodes as depicted in Figure 3.6. This can be problematic, since collocated sensor nodes often cooperate in observing a nearby physical event and thus may need a very small relative error. For example, estimates of the distance between the collocated nodes may include significant errors if the nodes use different paths. Local refinement procedures as described in [88] can somewhat improve the local consistency.

To achieve a reasonable precision, typically a large number of anchors is required, such that the maximum distance of any client node from a sufficient number of anchors is small. In [88], between 5% and 10% of all nodes, and in [89], between 10% and 20% of all nodes are anchors. An out-of-band mechanism is required to provide the anchors with precise location estimates. Such an out-of-band mechanism may present a serious drawback, since it typically implies additional hardware infrastructure, and special hardware must be attached to the sensor nodes. One typical example for such an out-of-band mechanism is GPS with its satellite infrastructure and resulting constraints, where anchor nodes must be equipped with expensive and energy-intensive GPS receivers.

In order to ensure that each node in the network has a sufficient number of neighbors which can act as references, the network must have a certain minimum density. This also implies that nodes at the edge of the network (with a lower

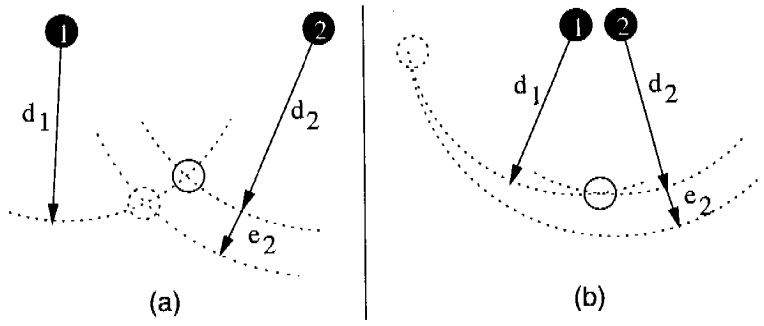


Figure 3.7: Localization error depends on the constellation of reference nodes.

number of neighbors) typically experience a reduced precision. Network density is particularly important if the collected constraints are loose, since then many constraints are needed to achieve precise location estimates. In [88], for example, each node must have an average of 7 to 12 neighbors on average in order to achieve a reasonable precision.

The arrangement of the anchors is also of importance for the achieved precision of localization in spacetime. Obviously, anchors should be evenly distributed across the sensor network in order to ensure that any node has a sufficient number of anchors in its vicinity. However, also the relative arrangement of anchors with respect to each other has an influence on the localization accuracy. For example, collinear anchors (i.e., anchor nodes that fall on a line in 3D space) and also approximately collinear anchors result in significantly reduced precision for localization in space. This is illustrated for localization in 2D in Figure 3.7. In part (a), the error e_2 in distance measurement results in a small error in the estimated location (dotted circle) w.r.t. the actual location of the node (solid circle). In part (b), where anchor nodes are almost collinear (i.e., fall on a point in 2D), the same error e_2 results in a much larger error in the estimated location.

3.4.2 Energy and Other Resources

As noted in Section 3.2.5, it is quite common that applications do only require a very limited scope and lifetime of localization, where actual scope and lifetime requirements depend on the occurrence of events in the physical environment. Hence, a significant amount of resources and energy could be saved if localization is only performed where and when needed and with the required precision.

However, distributed algorithms for localization in spacetime are often not well suited for on-demand localization. This is due to two main reasons. Firstly, localization of a single node typically requires the cooperation of many other nodes to act as references for obtaining constraints. For on-demand localization, (recursively) providing a sufficient number of reference nodes on-demand would be needed. However, managing this process is a complex task. For example, as noted in the previous section, the number and relative arrangement of the anchors must be considered by such mechanisms, as this is crucial for the achieved precision. More-

over, such selective localization may induce significant management overheads.

Secondly, many algorithms require a significant amount of convergence time for achieving the requested precision. For example, [60] reports a convergence time of 10 minutes in a network of only few tens of nodes. Hence, if a node requests localization, a significant amount of time will elapse before a location estimate with sufficient precision can be provided.

3.4.3 Network Dynamics

In Section 2.6.2 we discussed various effects of network dynamics found in sensor networks. These effects may have a significant impact on the performance and applicability of localization algorithms.

An important implicit assumption of many localization algorithms is that *before* the location of a node can be estimated, the node must obtain constraints involving a sufficient number of reference nodes. These references in turn must also be able to obtain constraints from a sufficient number of other reference nodes, and so on. Overall, in order to locate a node, there must be "constraint paths" from a client node to a sufficient number of anchors. This typically means that a sufficient portion of the network must be connected before localization can be performed.

However, a number of application projects (e.g., [48]) explore settings, where sensor nodes are mobile and network connectivity is sporadic. In such settings, there may not be a network connection between nodes that require localization in spacetime with respect to a common coordinate system. Algorithms with the above implicit connectivity assumption cannot be used for such applications. We will address this problem in Chapter 4.

As noted in the previous section, many algorithms take a significant amount of convergence time before delivering the requested precision. It is typically assumed that the network remains stable during the execution of the algorithm. However, node mobility and other effects of network dynamics may invalidate this assumption. This may lead to significantly increased convergence times (see, e.g., [98]), or may also prevent the algorithm from converging at all. In the latter case, the effective precision may reduce significantly.

Some algorithms construct an explicit overlay topology as noted in Section 3.3.4. Network dynamics may break these topologies or make them inefficient. Maintaining these topologies under a high degree of network dynamics may become an unacceptable resource overhead.

3.4.4 Configuration

Localization algorithms may require a number of configuration parameters whose values differ from node to node. In order to allow unattended operation, configuration must be performed with little or no support from human operators.

Typical examples for configuration parameters are the set of reference nodes to use, network link calibration parameters (e.g., minimum delay of a wireless link),

and sensor calibration parameters (e.g., for distance measurements). While some of these parameters can be automatically configured and adapted, this is not so easy for other parameters. For example, calibration may be a particularly tricky issue in sensor networks, because typical low-cost sensors used on sensor nodes are very sensitive to environmental parameters such as temperature and humidity. If these sensors are exposed to a harsh and dynamic physical environment, the output of the sensors includes significant errors. In [108], for example, the authors observed an average error of approx. 75% for distance measurements with uncalibrated sensors based on time of flight of an ultrasound signal. Additionally, sensor orientation, wear, and dirt lead to systematic but dynamically changing errors. Hence, calibration parameters often cannot be statically configured, but must be dynamically updated to reflect the changing setup.

3.5 Summary

This chapter discussed issues related to time and space in wireless sensor networks. We presented a common framework that supports a unified view on space and time. In particular, we presented applications of space and time and classified these into three broader categories: applications related to interfacing a sensor network to an external observer, applications related to interfacing a sensor network to the observed real-world, and applications related to coordination among sensor nodes. We also discussed various requirements and general approaches that apply both to time synchronization and localization: internal vs. external localization/synchronization, local vs. global scales, the use of point estimates vs. bounds to represent points in time/space, the need for synchronized points vs. distances, scope and lifetime of synchronization/localization, precision, and other QoS requirements. Then we considered distributed algorithms for synchronization and localization and pointed out five common structural elements of many algorithms: bootstrapping, obtaining constraints, combining constraints, selecting constraints, and maintaining localization/synchronization over time. We then showed that these algorithms are affected by a number of limitations and trade-offs in four broader categories: with respect to anchor infrastructures, energy and resource consumption, network dynamics, and with respect to configuration.

Chapter 4

Time Synchronization

The significance of physical time for sensor networks has been reflected by the development of a number of time synchronization algorithms in the recent past. However, most of these approaches have been designed for "traditional" sensor networks, covering only a small region of the design space discussed in Section 2.2. We will identify an important region in the design space that is not sufficiently supported by existing approaches. In order to fill this gap, we present and evaluate an algorithm called "Time-Stamp Synchronization".

We begin our discussion in Section 4.1 by studying fundamental system models and by presenting concrete algorithmic techniques for synchronization. The discussion will be structured according to the common framework we developed in the previous chapter. Referring to these techniques, we present existing algorithms for time synchronization in Section 4.2. Our algorithm will be motivated and presented in Sections 4.3 and 4.4.

4.1 Background

This section reviews models and concepts for time synchronization. The discussion is structured according to Section 3.3.

4.1.1 Clock and Communication Models

In the following two subsections we discuss models of communication and of hardware clocks. Communication is fundamental for measuring temporal relationships among nodes. Hardware clocks are an important tool for maintaining synchronization over time.

Clock Models

Most computer systems in use today are based on clocked circuits and hence contain so-called *digital clocks*. Such hardware clocks are a valuable tool for time

synchronization, since they can be used to maintain synchronization over time as discussed in Section 3.3.5.

A typical hardware clock consists of a quartz-stabilized oscillator and a counter that is incremented by one every oscillation period (e.g., upon detection of a falling or rising edge). If the periodic time T of the oscillator is known, the counter h can be used to obtain approximate measurements of real-time intervals in multiples of T.

More formally, the clock counter displays value $h(t)$ at real time t and is incremented by one at a frequency of f. The rate of the counter is defined as $f(t) = \mathrm{d}h(t)/\mathrm{d}t$. An ideal digital clock would have a rate of 1 at all times. However, the periodic time of the oscillator and hence the clock rate depend on various parameters such as age of the quartz, supply voltage, environmental temperature and humidity. This so-called clock drift is formally defined as the deviation of the rate from 1 or

$$\rho(t) = f(1) - 1 \tag{4.1}$$

Since sensor nodes are typically operated under a well-defined ranges of the above parameters, it is reasonable to assume a maximum possible drift $\rho_{\max}$, such that

$$|\rho(t)| \leq \rho_{\max} \tag{4.2}$$

for all t. Typical values for $\rho_{\max}$ are 1ppm to 100ppm, where 1ppm $= 10^{-6}$.

Some researchers make additional assumptions about the clock rate such as bounded drift variation or constant clock rate. The latter may not be a reasonable model for sensor networks that are exposed to significant changes of environmental parameters.

Communication Models

Obtaining temporal constraints (cf. Section 3.3.2) is typically implemented by communication among sensor nodes. There are three major characteristics of communication that affect time synchronization.

Firstly, a communication network has the multicast/broadcast capability if a single message can be received by multiple/all nodes within the communication range of the sender. Since commonly used sensor nodes use radio communication, message broadcasts are typically supported. Broadcasts can, for example, be exploited to synchronize an arbitrary number of nodes with a fixed number of messages.

Secondly, a communication link is said to be symmetrical if it can be used in both directions. A link is asymmetrical if it can be used in one direction only. For example, some algorithms rely on the ability to measure round-trip-delays, which requires symmetrical links (cf. Section 2.6.2).

Thirdly, the latency or delay characteristics of a communication link are of utmost importance for time synchronization. Known constant delays can be easily compensated by including a respective constant offsets in time synchronization

algorithms. However, the latency of a communication link typically varies over time (cf. Section 2.6.2). These variable delays can be attributed to four major sources:

- Send time, lasting from when the application issues a "send" command to the operating system, until when a raw network message has been constructed that could be delivered to the communication medium. Variable delays result from operating system context switches, network protocol processing, and from hardware interrupts.

- Medium access time. The communication medium is typically shared by many network nodes, such that immediate access may not be possible, causing additional variable delays. For example, random backoff mechanisms are often used to resolve collisions. Since the channel bandwidth is rather low for sensor networks (few tens of kilobit per second), this delay component may vary between zero and few tens of milliseconds.

- Propagation time. The time it takes for the radio signal to travel from the sender to the receiver depends on the distance between the nodes. These delays are often negligible for radio communication due to the high propagation speed and since the communication range is often rather small (few tens of meters). Typical values are between zero and few tens of nanoseconds.

- Receive time, lasting from the arrival of the signal at the antenna of the receiver, until when the application is notified about the arrival of the message. There is often a variable delay in the order of the duration of one bit until the radio hardware triggers a receive interrupt to the processor. As for the send time, additional variable delays are caused by context switches, network protocol processing, and interrupts.

There are two important approaches to eliminate or to limit the impact of these variable delays. The first approach is to implement time synchronization in the MAC layer, thus eliminating send and medium access time, as well as most of the receive time. Another approach is to synchronize a set of nodes by sending a single broadcast message to them, such that all nodes experience identical send time and medium access time.

4.1.2 Obtaining Constraints

As discussed in Section 3.3.2, a client node that is to be synchronized has to obtain temporal constraints on its local time with respect to a time scale defined by a reference node. Such a constraint typically involves the state S of the reference node and a temporal relationship Δ between the client node and the reference node. For time synchronization, S typically consists of the local time h_r of the reference node, and Δ is a delay for sending a message from the reference node to the client node or vice versa. The client node can also refer to its current local time h_c that

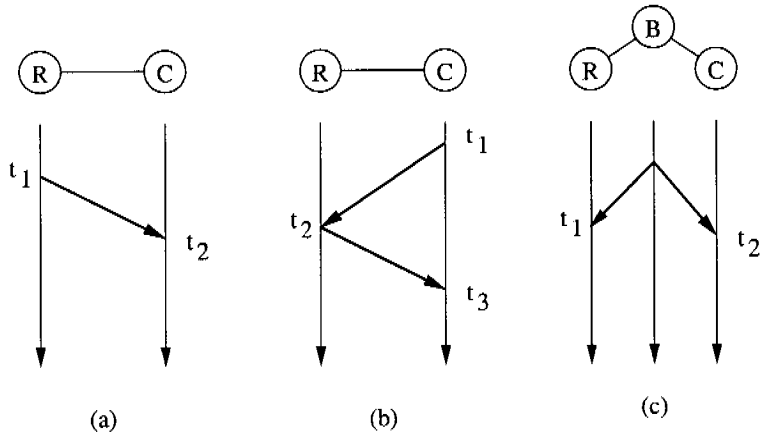


Figure 4.1: Obtaining temporal constraints. (a) Unidirectional, (b) round-trip, and (c) reference broadcast.

is defined by its (unsynchronized) hardware clock. The goal is to derive constraints on the synchronized time h'_c of the client node.

Figure 4.1 illustrates three basic approaches for obtaining such constraints. In (a), the reference node R sends a single message to the client node C at real time t_1, which is received by the client node at t_2. Note that nodes do not have access to real time t, only $h_r(t_1)$ and $h_c(t_2)$ (i.e., the values of the local clock counters at the respective real-time instants) is known to them. If the message delay $d = t_2 - t_1$ were known, a valid constraint would be $h'_c(t_2) = h_r(t_1) + d$. Here, $h_r(t_1)$ would represent the state information S of the reference node, and d would represent the temporal relationship Δ. However, d is unknown to both reference and client nodes. If upper and lower bounds $d_{\min} < d < d_{\max}$ are known, the following would be possible constraints:

$$h'_c(t_2) > h_r(t_1) \tag{4.3}$$
$$h'_c(t_2) > h_r(t_1) + d_{\min} \tag{4.4}$$
$$h'_c(t_2) < h_r(t_1) + d_{\max} \tag{4.5}$$
$$h'_c(t_2) = h_r(t_1) \tag{4.6}$$
$$h'_c(t_2) = h_r(t_1) + d_{\min} \tag{4.7}$$
$$h'_c(t_2) = h_r(t_1) + d_{\max} \tag{4.8}$$
$$h'_c(t_2) = h_r(t_1) + (d_{\min} + d_{\max})/2 \tag{4.9}$$

Note that constraints 4.3-4.5 are correct bounds, while constraints 4.6-4.9 are only approximations. Perhaps the most commonly used constraint is 4.6, because it does not require knowledge of bounds and is a good approximation if MAC layer techniques are used to eliminate send, medium access, and receive times. Note that this technique can be used to synchronize an arbitrary number of clients with one broadcast message.

In Figure 4.1 (b), the client node first sends a request message to the reference, which is immediately answered with a reply (an additional delay between receipt of the request and sending of the reply can be easily measured and compensated). Here, some notable constraints are:

$$h'_c(t_3) > h_r(t_2) \quad (4.10)$$
$$h'_c(t_3) > h_r(t_2) + d_{\min} \quad (4.11)$$
$$h'_c(t_3) < h_r(t_2) + (h_c(t_3) - h_c(t_1)) \quad (4.12)$$
$$h'_c(t_3) = h_r(t_2) + (h_c(t_3) - h_c(t_1))/2 \quad (4.13)$$

Note that constraints 4.10 and 4.12 give upper and lower bounds on $h'_c(t_3)$ without knowledge of $d_{\min}$ and $d_{\max}$. Constraint 4.13 is a good approximation if both messages have about the same delay (i.e., $t_2 - t_1 \approx t_3 - t_2$). Note that the number of required messages grows linearly with the number of clients.

In Figure 4.1 (c), an additional beacon node B sends a broadcast to both the reference and the client node. A commonly used constraint with this approach is

$$h'_c(t_2) = h_r(t_1) \quad (4.14)$$

This is a good approximation, since both nodes will receive the message almost concurrently as explained in Section 4.1.1. Note that exact bounds cannot be derived unless bounds on the message delay are known a priori. A single broadcast message can be used to synchronize an arbitrary number of clients, but additional message exchanges are required to transmit $h_r(r_1)$ from the reference to the client(s).

4.1.3 Combining Constraints

Although a single constraint may be sufficient to derive synchronized time h'_c at the client node, multiple constraints can significantly improve the achieved precision. Let us first consider the case where multiple constraints are available on $h'_c(t_0)$ for some real-time instant t_0. This might for example be achieved by concurrently requesting synchronization with multiple reference nodes. As discussed in the previous section, these constraints are either bounds or approximations.

If bounds are used, each pair of lower and upper bound can be considered as an interval. The intervals are open at one end if only lower or upper bounds are available. The combined interval is then computed as the intersection of all available intervals. If some or all intervals are closed, then this intersection may be empty. Hence, outlier rejection is often performed to eliminate intervals that would lead to an empty intersection. In general, different subsets of intervals result in a non-empty intersection. Various criteria can be applied to select such a subset, for example, minimizing the number of rejected intervals, or minimizing the length of the intersection interval.

If approximations are used, the average of all constraints can serve as the combined constraint. Outlier rejection can also be performed in order to reject false

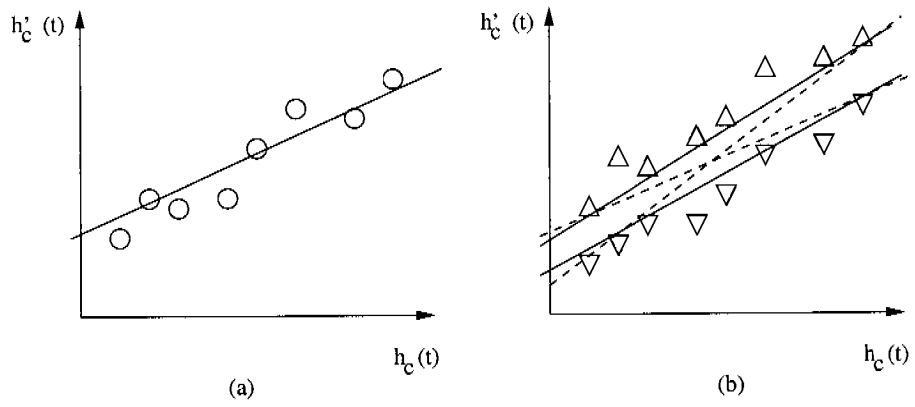


Figure 4.2: Prediction of synchronized time using a hardware clock. (a) Point estimates, (b) bounds.

constraints. One criterion for this would be to remove all constraints that deviate from the average by more than a certain threshold.

Sometimes it is possible to derive a confidence value for each constraint. The higher the confidence value, the better the constraint. With the round-trip technique in Figure 4.1 (c), for example, the inverse of the round-trip time $1/(h_c(t_3) - h_c(t_1))$ could be used as a confidence for the approximation $h'_c(t_3) = h_r(t_2) + (h_c(t_3) - h_c(t_1))/2$. If such confidence values are available, a weighted average can be computed. The confidence values can also be used to control outlier rejection.

4.1.4 Maintaining Synchronization

In the previous section we described how constraints on $h'_c(t_x)$ can be obtained for a real-time instant t_x. These constraints directly result in a estimate for the synchronized time of the client node at real time t_x. In order to obtain such an estimate for another instant t_y, the procedure of obtaining and combining constraints can be repeated as discussed in Section 3.3.5. However, there are two problems with this approach. Firstly, as discussed in Section 3.3.5, such an approach may exhibit significant overheads in terms of communication and computation. Secondly, the application may request synchronized time at t_x, but due to processing delays the constraints refer to $t_x + \epsilon$. In Figure 4.1 (a), for example, after the client application requests synchronized time at t_0, the client node must wait until it receives a synchronization message from the reference node in order to obtain constraints on $h'_c(t_2)$, where $t_2 > t_0$.

In order to approach these problems, the client node may use its local hardware clock to predict $h'_c(t_y)$, where no measured constraints are available for real time t_y. For this, a mapping from $h_c(t)$ to $h'_c(t)$ must be derived, given a set of measured constraints on $h'_c(t_{x_i})$ for different real-time instants t_{x_i} as illustrated in Figure 4.2. In (a), each circle indicates a data point $(h_c(t_{x_i}), h'_c(t_{x_i}))$ where a point estimate for $h'_c(t_{x_i})$ is known. In (b), lower ($\bigtriangledown$) and upper bounds ($\bigtriangleup$) for $h'_c(t_{x_i})$ are known.

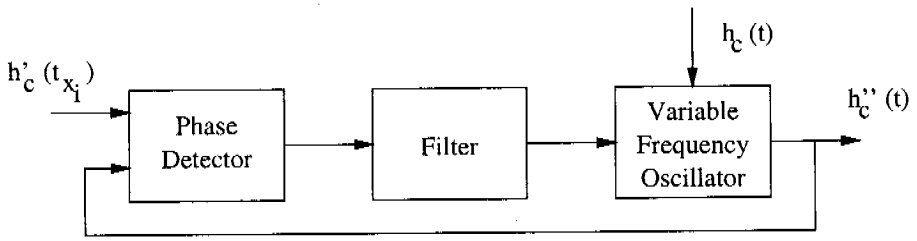


Figure 4.3: Using a phase-locked loop to establish a mapping between hardware clock and synchronized time.

Various methods can be applied to derive the desired mapping of $h_c(t)$ to $h'_c(t)$, given the above sets of constraints. A commonly used approach is to assume a linear relationship

$$h'_c(t) = \alpha h_c(t) + \beta \tag{4.15}$$

Note that α can be interpreted as the relative drift or rate difference between synchronized time and the hardware clock, and β indicates the offset between synchronized time and the hardware clock. Once α and β are known, the linear function 4.15 can be used to derive the current (predicted) synchronized time using the current hardware clock reading.

Line fitting techniques such as linear regression can be used to obtain estimates for α and β as illustrated by the line in Figure 4.2 (a). If bounds are used, a steepest and a flattest line can be fitted as illustrated by the dashed in lines in (b). The resulting values for α and β for the two lines can then be used as lower and upper bounds for rate and offset differences between synchronized time and the hardware clock. Alternatively, the line defined by the average of the α and β values of the two lines can be used. Yet another approach is the use of convex hulls, where two lines are fitted such that they cut off minimal half planes that contain all upper and all lower bounds, respectively. This is illustrated by the solid lines in Figure 4.2 (b). Note that the latter technique can also be applied if only lower or upper bounds are available.

An important question with the above techniques is how many data points (i.e., constraints) should be included in the line fitting procedure. With small numbers, few outliers can have a significant impact on the fitting result. This can be diluted by including a larger number of data points, which, however, results in increased memory footprint.

In practice, the relationship between synchronized time and hardware clock is often not linear. By repeating the line fitting procedure frequently, a piecewise linear approximation of that nonlinear relationship can be achieved. This approximation is the better, the fewer data points are included in each fitting procedure. Often it is mandatory to ensure that this piecewise approximation is continuous, which may require the introduction of additional constraints on the fitted lines.

A different approach to derive a mapping of $h_c(t)$ to $h'_c(t)$ is the use of phase-locked loops (PLL). As depicted in Figure 4.3, a PLL consists of at least three

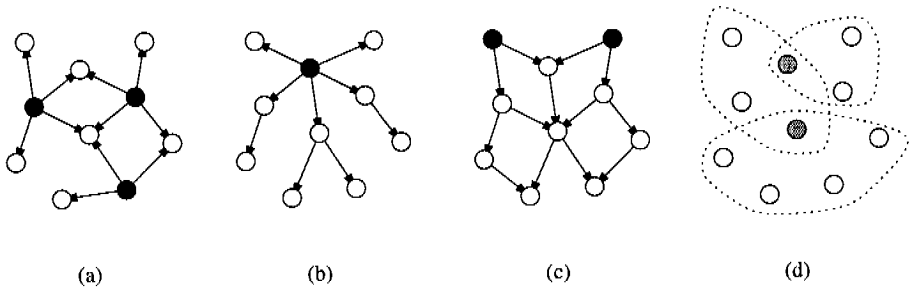


Figure 4.4: Commonly used overlay topologies. (a) Stars, (b) tree, (c) hierarchy, and (d) clusters.

components. A variable frequency oscillator (VFO) is a component that produces an estimation $h_c''(t)$ for synchronized time $h_c'(t)$. The frequency $\mathrm{d}h_c''(t)/\mathrm{d}t$ can be controlled by an input signal of the VFO. The generation of the output signal is usually based on the hardware clock signal $h_c(t)$, which therefore represents the second input to the VFO. The phase detector (PD) estimates the phase offset of its two input signals and generates an output signal that is proportional to the observed phase difference. The first input of the PD is the sequence of estimates $h_c'(t_{x_i})$, the second input is the output $h_c''(t)$ of the VFO. The output of the PD is fed to a filter that integrates the observed phase differences over time and produces an output signal to control the frequency of the VFO. Overall, a PLL can be considered as a feedback loop that adjusts the frequency and phase of the VFO to the discrete input signal $h_c'(t_{x_i})$. Hence, the output $h_c''(t)$ of the VFO is an approximation of synchronized time $h_c'(t)$.

In contrast to the line fitting approach, a PLL requires only minimal state information and does not assume a linear relationship. However, due to the delay introduced by the filter, it may take minutes before the VFO output converges to a stable frequency.

Note that both of the above approaches are a combination of prediction and dead reckoning (cf. Section 3.3.5). The hardware clock can be considered a "time sensor" that is calibrated using the observed past behavior of synchronized time.

4.1.5 Selecting Constraints

As discussed in Section 3.3.4, there are two basic approaches to control the selection of reference nodes by a client node: *structured* and *unstructured* approaches. With structured approaches, an overlay topology is constructed and maintained, which can be interpreted as a subgraph of the network. A client node then only considers neighbors in this overlay topology as potential references. With unstructured approaches, a client node may consider any of its neighbors as references.

As depicted in Figure 4.4, commonly used overlay topologies are stars, trees, hierarchies, and clusters. With stars, a number of anchor nodes (black) are distributed across the network, such that each client node (white) has one or more anchors in its one-hop neighborhood. The anchors are synchronized with each

other by means of an out-of-band mechanism. With trees, a single anchor node forms the root of a spanning tree. The nodes in the tree synchronize top-down with their parents, such that each node has exactly one reference node. Hierarchies are directed acyclic graphs with one or more anchors at the top. Nodes synchronize top-down with their parents. With clustering, nodes are grouped into clusters, such that adjacent clusters share common gateway nodes (gray). The nodes in a cluster are synchronized in some way with each other. Gateway nodes independently participate in the synchronization process of each adjacent cluster, such that they can translate between the time scales of the clusters they participate in.

4.2 Related Work

This section presents and discusses existing approaches for time synchronization. We first discuss related and influential approaches from other domains, pointing out their shortcomings in the context of sensor networks. Our main focus is, however, on algorithms which have been specifically developed for sensor networks. These are based on the models and concepts presented in the previous section.

4.2.1 Logical Time

Logical time [54, 64] provides mechanisms to time-stamp events (e.g., receipt or sending of a message). For example, $C(e_1)$ is the logical time stamp of event e_1. If a second event e_2 is causally dependent on e_1 (i.e., $e_1 \rightarrow e_2$), then $C(e_1) < C(e_2)$ will hold with respect to some ordering relation "$<$" on time stamps. Two events are causally dependent if they happened in the same process or if there is a message path connecting e_1 and e_2.

In sensor networks, where sensor events are triggered by real-world phenomena, sensor events are only causally related if they have been generated by the same sensor node. Hence, temporal reasoning on events originating from different nodes cannot be supported by logical time. Moreover, it is often necessary to measure the amount of real-time elapsed between two sensor events, which cannot be achieved with logical time. Therefore, physical time must be used to relate events in the physical world.

4.2.2 Offline Time Synchronization

Time synchronization algorithms typically perform synchronization during the execution of a distributed application. Offline time synchronization performs synchronization only after a distributed application has finished execution. For this, each node logs events (e.g., message sent, message received) during the execution of the application. Each logged event is tagged with the time of occurrence using the unsynchronized hardware clock of the node. This results in an event log for each node. These logs are than provided as input to an offline time synchroniza-

tion algorithm. For each pair of nodes, the algorithm then computes a coordinate transformation between the local timescales of the nodes.

Such event logs are collected over extended periods of time (e.g., hours). Existing algorithms [5, 31] typically assume that the rate difference between the hardware clocks and the message delays are constant during the logging period.

In the context of sensor networks, offline approaches are of limited applicability, since synchronized time is often required during the execution of an application. Secondly, both message delays and clock rates may be subject to significant variation in sensor networks (cf. Section 4.1.1).

4.2.3 Network Time Protocol (NTP)

NTP [68] has been designed for large-scale networks with a rather static topology (such as the Internet). Nodes are externally synchronized to a global reference time that is injected into the network at many places via a set of master nodes, so-called "stratum 1" servers. These master nodes are synchronized out of band, for example via GPS. Nodes participating in NTP form a hierarchy: leaf nodes are called clients, inner nodes are called stratum L servers, where L is the level of the node in the hierarchy. The parents of each node must be specified in configuration files at each node. A node in the hierarchy uses round-trip measurements to obtain bounds on synchronized time with respect to its parents. Outliers are removed from the resulting set of intervals. From the intersection of the remaining intervals, a point estimate of synchronized time is computed and used as input for a PLL that eventually outputs synchronized time.

While NTP has been successfully applied in the Internet, it fails to meet many of the technical challenges discussed in Section 2.6. Many of these limitations are closely related to the issues discussed in Section 3.4.

Resource and energy constraints. Resource constraints may preclude the use of GPS or other technologies for out-of-band synchronization of NTP master nodes. NTP is also not optimized for energy efficiency, simply because this is not an issue in traditional distributed systems. Energy overhead in NTP results from several sources. Firstly, the service provided by NTP typically cannot be dynamically adapted to the varying needs of an application. Hence, with NTP all nodes would be continuously synchronized with maximum precision, even though only subsets of nodes might occasionally need synchronized time with less-than-maximum precision.

Secondly, NTP uses the processor and the network in ways that would lead to significant overhead in energy expenditure in sensor networks. For example, NTP maintains a synchronized system clock by regularly adding small increments to the system-clock counter. This behavior precludes the processor from being switched to a power-saving idle mode. In addition, NTP servers must be prepared to receive synchronization requests at any point in time. However, constantly listening is an energy-wise costly operation in sensor networks; many sensor-network protocols

therefore switch off the radio whenever possible.

Network dynamics. The operation of NTP is largely independent of the underlying physical network topology. In the NTP overlay hierarchy, a master and a client can be separated by many hops in the physical network, even though they are neighbors in the overlay hierarchy. As discussed in Section 2.6.2, multi-hop paths may be very unstable and unpredictable in a sensor network. NTP, however, depends on the ability to accurately estimate the delay characteristics of multi-hop network links.

NTP implicitly assumes that network nodes that shall be synchronized are a priori connected by the network. However, this assumption may not hold in dynamic sensor networks with mobile nodes as discussed in Section 2.6.2.

Infrastructure. In order to improve the precision and availability of synchronization in large networks, reference time is injected at many points into the network. Hence, any node in the network is likely to find a source of reference time in a distance of only a few hops. However, such an approach requires an external infrastructure of reference-time sources which have to be synchronized with some out-of-band mechanism. Where this is not feasible, NTP would have to operate with a single master node, which uses its local time as the reference time. In large sensor networks, the average path length from a node to this single master is long, leading to reduced precision. If collocated nodes end up using different synchronization paths, they will be poorly synchronized (cf. Section 3.4.1).

Configuration. NTP requires the specification of one or more potential synchronization masters for each node. This is an appropriate solution for networks with a rather static topology, where configurations remain valid for extended periods of time. In sensor networks, however, network dynamics necessitate a frequent adaptation of configuration parameters.

4.2.4 Time Synchronization for Sensor Networks

In this section we review time synchronization approaches that have been specifically devised for sensor networks. Note that most of these algorithms have been proposed after our work presented in Section 4.4.

Reference-Broadcast Synchronization (RBS)

RBS [33] denotes some nodes as beacons which frequently broadcast messages to a set of client nodes that should be synchronized as illustrated in Figure 4.1 (c) on page 69. Clients that receive such a broadcast exchange their respective reception times to obtain mutual constraints. Each client collects multiple such constraints and uses linear regression to compute relative time offsets and rate differences to the other client nodes. The offset and rate difference between a pair of

client nodes defines a coordinate transformation between the local time scales (i.e., coordinate systems) of these nodes. To extend this scheme to multi-hop networks, the network is clustered such that a single beacon can synchronize all nodes in its cluster. Gateway nodes that participate in two or more clusters independently take part in the reference-broadcast procedure of all adjacent clusters. By knowing offsets and rate differences to nodes in all clusters, gateway nodes can compute coordinate transformations between all adjacent clusters. Time synchronization across multiple hops is then provided by transforming clock readings between the local time scales (i.e., coordinate systems) of the nodes.

In experiments it has been shown that adjacent Berkeley Motes can be synchronized with an average error of 11 μs by using 30 broadcasts. Over multiple hops, the average error grows with $O(\sqrt{n})$, where n is the number of hops.

Tiny-Sync and Mini-Sync (TS/MS)

Tiny-Sync and Mini-Sync [95] are methods for pairwise synchronization of sensor nodes. Both Tiny-Sync and Mini-Sync use multiple round-trip measurements and a line-fitting technique to obtain the offset and rate difference of the two nodes. For this, a constant clock rate is assumed. To obtain data points for line fitting, multiple round-trip measurements are performed as depicted in Figure 4.1 (b) on page 69. Each round-trip measurement is used to obtain lower and upper bounds on $h'_c(t)$. Then, the line-fitting technique depicted in Figure 4.2 (b) on page 71 is used to calculate two lines with minimum and maximum slope. Slope and axis intercept of these two lines then give bounds for the relative offset and rate difference of the two nodes. The line with average slope and intercept of the two lines is then used as the offset and rate difference between the two nodes.

Note that each of the two lines is unambiguously defined by two (a priori unknown) data points. The same results would be obtained if the remaining data points were eliminated. Since the computational and memory overhead depends on the number of data points, it is a good idea to remove as many data points as possible before the line fitting. Tiny-Sync and Mini-Sync only differ in this elimination step. Essentially, Tiny-Sync uses a heuristic to keep only two data points for each of the two lines. However, the selected points may not be the optimal ones. Mini-Sync uses a more complex approach to eliminate exactly those points that do not change the solution. Hence, Tiny-Sync achieves a slightly suboptimal solution with minimal overhead, Mini-Sync gives an optimal solution with increased overhead.

Measurements on a 802.11b network with 5000 data points resulted in an offset of 945 μs (3230 μs) and a rate difference of 0.27 ppm (1.1 ppm) for adjacent nodes (nodes five hops away).

Lightweight Time Synchronization (LTS)

LTS [101] is a synchronization technique that provides a specified precision with little overhead, rather than striving for maximum precision.

Two algorithms are proposed: one that operates on demand for nodes that actually need synchronization, and one that proactively synchronizes all nodes. Both algorithms assume the existence of one or more master nodes that are synchronized out-of-band to a reference time. The proactive algorithm proceeds by constructing spanning trees with the masters at the root by flooding the network. In a second phase, nodes synchronize to their parents in the tree by means of round-trip synchronization. The synchronization frequency is calculated from the requested precision, from the depth of the spanning tree, and from the drift bound $\rho_{\max}$.

The on-demand version also assumes the existence of one or more master nodes. When a node needs synchronization, it sends a request to one of the masters using any routing algorithm (this is not further specified). Then, along the reverse path of the request message, nodes synchronize using round-trip measurements. The synchronization frequency is calculated as in the proactive version described above. In order to reduce synchronization overhead, each node may ask its neighbors for pending synchronization requests. If there are any such requests, the node synchronizes with the neighbor, rather than executing an independent multi-hop synchronization with a reference node.

The overhead of the algorithms was examined in simulations with 500 nodes uniformly placed in a 120 m $\times$ 120 m area, a target precision of 0.5 s, and a duration of 10 hours. The centralized algorithm performed an average of 36 pairwise synchronizations per node. The distributed algorithm executed 4-5 synchronizations on average per node if 65% of all nodes request synchronization.

Timing-Sync Protocol for Sensor Networks (TPSN)

TPSN [38] provides synchronization for a whole network. First, a node is elected as a synchronization master (details for this are not specified), and a spanning tree with the master at the root is constructed by flooding the network. In a second phase, nodes synchronize to their parents in the tree using round-trip measurements. Synchronization is performed in rounds and initiated by the root broadcasting a synchronization-request message to its children. Each child then performs a round-trip measurement to synchronize with the root. Nodes further down in the tree overhear the messages of their parents and start synchronization when their parents have synchronized. To eliminate message-delay uncertainties, time-stamping for the round-trip measurements is done in the MAC layer. In case of node failures and topology changes, master election and tree construction must be repeated.

Measurements showed that two adjacent Berkeley Motes can be synchronized with an average error of 16.9 μs, which is a worse figure than the 11 μs given for RBS in [33]. However, the authors of [38] claim that a re-implementation of RBS on their hardware resulted in an average error of 29.1 μs between adjacent nodes, effectively claiming that TPSN is about twice as precise as RBS.

TSync

TSync [26] provides two protocols for external synchronization: the Hierarchy Referencing Time Synchronization Protocol (HRTS) for proactive synchronization of the whole network, and the Individual-Based Time Request Protocol (ITR) for on-demand synchronization of individual nodes. Both protocols use an independent radio channel for synchronization messages in order to avoid inaccuracies due to variable delays introduced by packet collisions. In addition, the existence of one or more master nodes with access to a reference time is assumed.

With HRTS, a spanning tree with the master at the root is constructed. Then, the master uses the reference broadcasting technique illustrated in Figure 4.1 (c) on page 69 to synchronize its children. Each child node now repeats the procedure for its subtree.

Measurements in a network of MANTIS sensor nodes showed a mean synchronization error of 21.2 μs (29.5 μs) for two adjacent nodes (nodes three hops away). For comparison, RBS was also implemented, giving an average error of 20.3 μs (28.9 μs).

Interval-Based Synchronization (IBS)

Interval-based synchronization was first proposed in [63], where a bounded-drift model is assumed. The network nodes perform external synchronization by maintaining a lower and upper bound on the current time. During communication between two nodes, the bounds are exchanged and combined by choosing the larger lower and the smaller upper bound. This amounts to intersecting the time intervals defined by each pair of bounds. Between communications, each node advances its bounds according to the elapsed real time and the known drift bounds. In [91], the model was refined by including bounded drift variation and fault-tolerance.

In [15], the simple approach from [63] was shown to be worst-case-optimal, where the worst case is the one where all clocks run with maximal drift. A considerable improvement in the synchronization quality can be achieved by having each node store, maintain, communicate, and use the bounds from its last communications with other nodes. In [65], it was shown that optimal interval-based synchronization can only be achieved by having nodes store and communicate their entire history. Obviously, this becomes prohibitive with growing network size and lifetime. In realistic settings, the value of a piece of history data decreases rapidly with its age. Therefore, efficient average-case-optimal synchronization can be obtained by using only recent data.

Flooding Time-Synchronization Protocol (FTSP)

FTSP [60] can be used to synchronize a whole network. The node with the lowest node ID is elected as the anchor whose local time serves as a reference for synchronization. If this node fails, then the node with the lowest ID in the remaining network is elected as the new anchor. The anchor periodically broadcasts

a synchronization message that contains its current local time. Nodes which have not received this message yet use the message contents to derive a constraint and broadcast the message to its neighbors. Each node collects eight such constraints and uses linear regression on these eight data points to estimate time offset and rate difference to the anchor. The algorithm is repeatedly executed to maintain synchronization over time. Time-stamping is performed in the MAC layer to minimize delay variability.

Measurements were performed in an eight-by-eight grid of Berkeley Motes, where each Mote has a direct radio link to its eight closest neighbors. With this setup, the network synchronized in 10 minutes to an average (maximum) synchronization error of 11.7 μs (38 μs), giving an average error of 1.7 μs per hop.

Asynchronous Diffusion (AD)

AD [56] supports the internal synchronization of a whole network. The algorithm is very simple: each node periodically sends a broadcast message to its neighbors, which reply with a message containing their current time. The receiver averages the received time stamps and broadcasts the average to the neighbors, which adopt this value as their new time. It is assumed that these operations are atomic, that is, the averaging operations of the nodes must be properly sequenced.

Simulations with a random network of 200 static nodes showed that the synchronization error decreases exponentially with the number of rounds.

Time Diffusion Synchronization (TDP)

TDP [98] supports the synchronization of a whole network. Initially, a set of master nodes is elected. For external synchronization, these nodes must have access to a global time. This is not required for internal synchronization, where masters are initially unsynchronized.

Master nodes then broadcast a request message containing their current time, and all receivers send back a reply message. Using these round-trip measurements, a master node calculates and broadcasts the average message delay and standard deviation. Receiving nodes record these data for all leaders. Then, the receivers turn themselves into so-called "diffused leaders" and repeat the procedure. The average delays and standard deviations are summed up along the path from the masters. The diffusion procedure stops at a given number of hops from the masters.

All nodes have now received from one or more masters m the time $h_m(t_0)$ at the initial leader, the accumulated message delay Δ_m, and the accumulated standard deviation β_m. A clock estimate is computed as $\sum_m \omega_m(h_m(t_0) + \Delta_m)$, where the weights ω_m are inversely proportional to the standard deviation β_m. After all nodes have updated their clocks, new masters are elected and the procedure is repeated until all node clocks have converged to a common time.

In a simulation with 200 static nodes with 802.11 radios and a delay of 5 seconds between consecutive synchronization rounds, the deviation of time across the network dropped to 0.6 seconds after about 200 seconds.

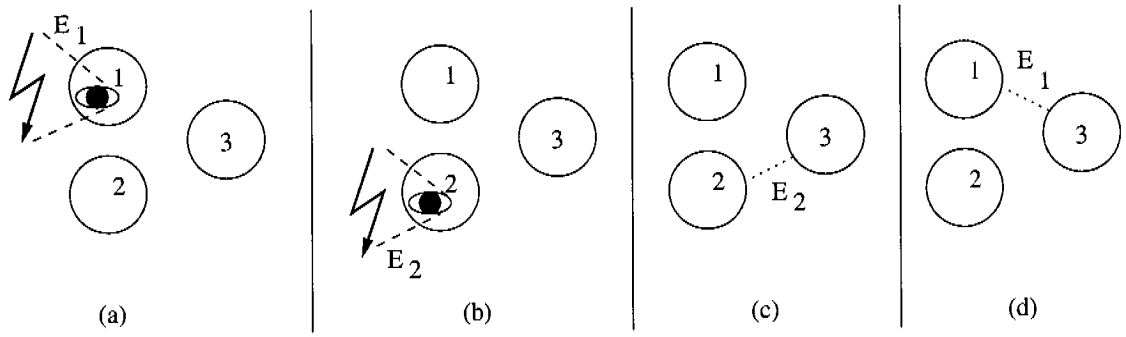


Figure 4.5: Message transport across partition boundaries. (a-b) Sensor nodes 1 and 2 collect sensor readings while disconnected from the network. (c-d) Later, sensor nodes 1 and 2 report their findings to node 3 for data fusion. At no point in time there is a network connection between node 1 and 2.

4.3 Problem Statement

One of the main uses of synchronized time in wireless sensor networks is time-stamping of events to support data evaluation, aggregation, and fusion as mentioned in 3.1. For this, a sensor node i generates a time stamp $S_i(t_E)$ that represents the real-time instant t_E when event E occurred. This time stamp may be included in network messages along with other parameters that describe the observed event. When a sensor node receives many such time stamps from one or more sensor nodes (including itself), these time stamps should refer to a common time scale, such that time stamps can be compared, ordered, etc.

One possible approach to the above problem is the use of time synchronization among all nodes of the network. Then, time-stamping can be implemented by setting $S(t_E) := h'(t_E)$. One of the time synchronization algorithms for sensor networks discussed in Section 4.2 could then be used to provide synchronized time. However, we will show below that this approach cannot support a relevant class of applications. Our goal is the development of a time-stamping approach for the region in the design space we will characterize below.

4.3.1 Intermittent Connectivity

None of the algorithms discussed in Section 4.2 can support networks with intermittent or sparse connectivity, where messages are relayed across temporary partitions by mobile nodes as discussed in Section 2.6.2.

Consider for example the ZebraNet application discussed in Section 2.4.1, where nodes are attached to wild animals, forming a network with sporadic connectivity. Figure 4.5 depicts, from left to right, four snapshots of such a network. At real-time t_1 node 1 detects some event. At t_2 node 2 detects another event. At t_3 node 2 passes by the mobile base station (node 3), a communication link is established and E_2 is sent to the base station. At t_4 node 1 passes by the base station, a link is established and E_1 is sent to the base station.

Now the base station wants to determine a temporal relationship among E_1 and E_2, for example, it might be necessary to determine whether E_1 happened after E_2, or the time between the occurrence E_1 and E_2 might be of interest.

Note that there has not been a network connection between nodes 1 and 2 before the occurrence of the events E_1 and E_2, so the clocks of the two nodes cannot be synchronized with each other in advance to provide synchronized time for time stamping.

4.3.2 Resource Efficiency

In many applications, relevant events occur rarely. In the bird monitoring application described in Section 2.4.1, a relevant event could be a bird leaving or entering its burrow. Hence, synchronized time is only needed at rare occasions, namely where and when a relevant event occurs. A cross-layer approach (cf. Section 2.7.6) to time synchronization that provides synchronized time on demand only where and when needed can be expected to perform significantly better than using a general-purpose synchronization algorithm that runs independently of the time-stamping in the application layer.

4.3.3 Precision for Collocated Nodes

Many physical phenomena have a rather local geographical scope. Typically, only geographically collocated nodes have to cooperate in order to monitor such a phenomenon. These collocated nodes may require very precisely synchronized time in order to aggregate or fuse sensory data. As discussed in Section 3.4.1, the precision of anchor-based algorithms is determined by the placement of the anchors, such that collocated nodes may end up with an imprecise mutual synchronization. A localized, anchor-free algorithm (cf. Section 2.7.3) can be expected to give better precision for collocated nodes.

4.3.4 Correctness

Data fusion is often very sensitive to even small synchronization errors. Correct ordering of events, for example, may be wrong if the synchronization error is larger than the time between the occurrence of two events. A time synchronization algorithm that can provide guaranteed bounds on time stamps would be helpful for such applications.

4.4 Time-Stamp Synchronization

In this section we present an algorithm called "Time-Stamp Synchronization" (TSS), which is suitable for the time-stamping problem stated in the previous section. This algorithm enables participating nodes to reason about sets of time

stamps (e.g., determine temporal ordering and time spans) received from arbitrary nodes even in the presence of intermittent network connectivity.

We will consider message flows in ad hoc sensor networks, which can be depicted by (time-independent) message flow graphs, where the nodes of the graph correspond to network nodes, each equipped with its own clock. Paths in the graph correspond to possibly delayed message flows between the nodes (i.e., connectivity between the nodes may be intermittent). Without loss of generality, we will only consider linear graphs as depicted in Figure 4.7. In case a message is broadcast to many nodes, the resulting graph can be considered as the union of many such linear graphs.

4.4.1 Algorithm Overview

The basic idea of the algorithm is *not* to synchronize the local clocks of the nodes, but instead generate time stamps using unsynchronized local clocks. When such "local" time stamps are passed between nodes as part of network messages, they are transformed to the local time scale of the receiving node.

Time stamps are represented as intervals. With each transformation step, the uncertainty (i.e., length) of these intervals increases due to clock drift and delay uncertainties (cf. Section 4.1.1). In particular, each transformation between the time scales of sender and receiver will consist of two steps: a transformation from sender time scale to real time, and a transformation from real time to the receiver time scale.

Time-stamp transformation is achieved by determining the age of each time stamp from its creation to its arrival at a sensor node. On a multi-hop path, the age is updated at each hop. The time stamp can then be transformed to the receiver's local timescale by subtracting the age from the time of arrival. The age of a time stamp consists of two components: (1) the total amount of time the time stamp resides in nodes on the path, and (2) the total amount of time needed to send the time stamp from node to node. The first component is measured using the local, unsynchronized clocks of the nodes on the path. The second component is bounded by round-trip measurements.

The synchronization information can be piggybacked to existing messages in most cases. Therefore, the overhead of the algorithm can be expected to be rather low.

The remainder of this Section is structured as follows. In Section 4.4.2 we discuss assumptions of TSS. In Sections 4.4.3-4.4.6 the algorithm is presented in detail. In Section 4.4.7 implementation details of TSS are given. An evaluation of TSS can be found in Section 4.4.8. Possible improvements of TSS are discussed in Section 4.4.9.

4.4.2 Assumptions

TSS is based on a number of assumptions. Firstly, we assume that the hardware clocks of the sensor nodes have a bounded clock drift $\rho_{\max}$ (cf. Section 4.1.1). However, due to heterogeneous node hardware, we support different maximum drifts for different nodes. We will denote the maximum drift of node i with ρ_i (or $\rho_{\max}$ where i is obvious) throughout the chapter.

Secondly, we assume that payload message exchanges between adjacent nodes are acknowledged. This is typically the case due to the relatively high probability of message loss or corruption in sensor networks. Note that this implies that links between nodes remain established long enough to allow such a two-way message exchange. An explicit acknowledgment is not needed if the sender can overhear the receiver forwarding the message to the next hop, which is typically the case in broadcast networks.

Besides the above two mandatory preconditions, our algorithm can profit from two optional assumptions.

Firstly, the precision of TSS can be improved if messages can be time-stamped in the MAC layer immediately before the first bit of the message is delivered to the transmitter, and after the first bit has arrived in the receiver, both using the unsynchronized local hardware clock. If this feature is not available, time-stamping can be performed at the application level immediately before sending and after receiving a message. This, however, will result in reduced precision due to the reasons discussed in Section 4.1.1.

Secondly, the precision of TSS can be improved if a lower bound on the message delay is known. Due to network heterogeneity, different nodes may have different bounds. If time-stamping is performed at the application level, this bound may additionally depend on the size of the message. We will indicate the minimum delay for node i by D_i throughout the chapter, where D_i refers to the local time scale of the receiver node i of the message.

4.4.3 Time Transformation

As we will see in the following section, transforming real-time differences Δt into computer clock differences Δh and vice versa is at the heart of the algorithm. These transformations cannot be done exactly due to clock drift and message delay uncertainties as discussed in Section 4.1.1. Hence, the transformation of a time difference results in lower and upper bounds, or – in other words – in a slightly enlarged time difference.

From Equations 4.1 and 4.2 it follows immediately that

$$1 - \rho_{\max} \leq \frac{\Delta h}{\Delta t} \leq 1 + \rho_{\max} \tag{4.16}$$

which can be rearranged to give

$$\Delta t(1 - \rho_{\max}) \leq \Delta h \leq \Delta t(1 + \rho_{\max}) \tag{4.17}$$

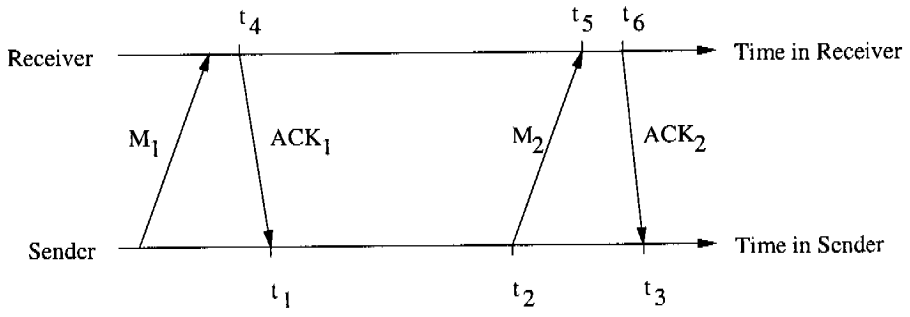


Figure 4.6: Message delay estimation using two consecutive acknowldedged message exchanges.

$$\Delta h/(1+\rho_{\max}) \leq \quad \Delta t \quad \leq \Delta h/(1-\rho_{\max}) \tag{4.18}$$

which means that we can approximate the computer clock difference Δh that corresponds to the real-time difference Δt by the interval $[(1-\rho_{\max})\Delta t,\ (1+\rho_{\max})\Delta t]$. Accordingly, the real-time difference Δt that corresponds to the computer clock difference Δh can be approximated by the interval $[\Delta h/(1+\rho_{\max}),\ \Delta h/(1-\rho_{\max})]$.

In order to transform a time difference ΔC from the local time of one node (with maximum drift ρ_1) to the local time of a different node (with maximum drift ρ_2), Δh is first estimated by the real-time interval $[\frac{\Delta h}{1+\rho_1},\ \frac{\Delta h}{1-\rho_1}]$, which in turn is estimated by the clock interval $[\Delta h\frac{1-\rho_2}{1+\rho_1},\ \Delta h\frac{1+\rho_2}{1-\rho_1}]$ with respect to the local time scale of node 2.

4.4.4 Message Delay Estimation

As pointed out earlier, the TSS algorithm determines bounds for the lifetime of a time stamp, which also includes the message delay d for sending the time stamp to a neighbor node. In Section 4.4.2 we assumed that such a message is acknowledged by the receiver. Thus, it is possible to measure the round-trip time rtt (time passed from sending the message in the sender to arrival of the acknowledgment in the sender) using the local clock of the sender. The message delay can then be estimated by the lower bound D_s and the upper bound rtt. Now the *sender* knows an estimation for the message delay, but in our algorithm the *receiver* has to know this approximation in order to transform the received time stamp. Passing the estimation from the sender to the receiver would take another pair of messages (one for passing the estimation from sender to receiver and an ack back to the sender), which would result in 100% message overhead.

Consider Figure 4.6, which shows two consecutive acknowledged message exchanges between a pair of sender and receiver. We want to estimate the message delay d for message M_2. Using the technique pointed out above the estimation would be

$$D_s \leq d \leq (h_s(t_3) - h_s(t_2)) - (h_r(t_6) - h_r(t_5))\frac{1-\rho_s}{1+\rho_r} - D_s \tag{4.19}$$

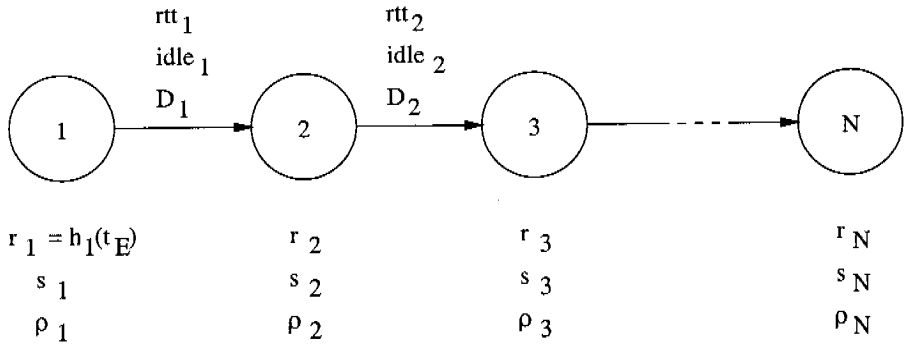


Figure 4.7: Message flow graph.

in terms of the sender's clock, where ρ_s and ρ_r are the ρ values, and h_s and h_r are the hardware clocks of sender and receiver, respectively. A different estimation is

$$D_{\mathrm{r}} \leq d \leq (h_r(t_5) - h_r(t_4)) - (h_s(t_2) - h_s(t_1))\frac{1-\rho_r}{1+\rho_s} - D_{\mathrm{r}} \qquad (4.20)$$

in terms of the receiver's clock that makes use of two consecutive message transmissions. The advantage of this estimation is that the receiver knows an estimation for d without additional message exchanges since $h_s(t_2) - h_s(t_1)$ can be piggybacked on M_2. We will call $h_r(t_5) - h_r(t_4)$ the round trip time *rtt* of the message, which is measured using the receiver's clock, and $h_s(t_2) - h_s(t_1)$ will be referred to as the idle time *idle* of the message, which is measured using the sender's clock.

However, the second estimation also has two disadvantages. The individual values for *rtt* and *idle* can become quite large if the nodes communicate rarely, which leads to bad estimations due to the clock drift of the local clocks. This problem can be relaxed by sending a dummy message if the resulting *idle* value for the message would be too large.

The second disadvantage stems from the fact that t_4, t_1 and t_5, t_2 are associated with different message transmissions, forcing both sender and receiver to keep track of state information between message transmissions (t_1 and t_4 in figure 4.6, respectively). This is problematic if a node sends messages to or receives messages from many different nodes over time. However, this problem can be mitigated by deleting state information at the cost of a later dummy message exchange to re-initialize the clock values, for example in a least-recently-used manner. Thus, one can trade off memory consumption for message overhead.

4.4.5 Time-Stamp Calculation

TSS consists of two major parts. First, a representation of time stamps and rules for transforming them when they are passed between nodes inside messages, and second, rules for comparing time stamps.

A time stamp for event E that occurred at real time t_E is represented in node i by the interval $[h_i^l, h_i^r]$. The end points of the interval refer to the time scale defined by the hardware clock h_i of node i. If the event occurred at time $h_i(t_E)$,

then we require that $h_i^l \leq h_i(t_E) \leq h_i^r$. In other words, $S_i(E)$ is an estimation of the unknown value $h_i(t_E)$.

Consider Figure 4.7, where node 1 passes a time stamp on to nodes $2, 3, ..., N$ along the depicted chain. Each node i has 3 attributes, the local time r_i when the message containing the time-stamp interval is received, the local time s_i when the message containing the time-stamp interval is sent again, and the maximum clock drift ρ_i. All values refer to the time scale defined by the local hardware clock. Each edge in the graph has three attributes: the round trip time rtt_i (referring to receiver's time scale), the idle time $idle_i$ elapsed after sending the last message over this edge (referring to sender's time scale), and the minimum delay D_i for sending the message (referring to receiver's time scale). Separate instances of the attributes rtt_i and $idle_i$ have to be maintained for each message, for simplicity we only consider a single multi-hop message transmission from node 1 to node N.

The generator of a time-stamped message is a special case, because it does not receive a message. Instead, r_1 is set to the occurrence time $h_1(t_E)$ of the event E (cf. Figure 4.7). Consider the time-stamp interval as it is being passed along the chain from node 1 to node N.

Node 1

$$[r_1\ ,\ r_1] = [h_1(t_E)\ ,\ h_1(t_E)] \tag{4.21}$$

Node 2

$$\left[r_2 - (s_1 - r_1)\frac{1+\rho_2}{1-\rho_1} - (rtt_1 - D_1 - idle_1\frac{1-\rho_2}{1+\rho_1})\ ,\right.$$
$$\left. r_2 - (s_1 - r_1)\frac{1-\rho_2}{1+\rho_1} - D_1\right] \tag{4.22}$$

Node 3

$$\left[r_3 - (s_1 - r_1)\frac{1+\rho_3}{1-\rho_1} - (s_2 - r_2)\frac{1+\rho_3}{1-\rho_2}\right.$$
$$-((rtt_1 - D_1)\frac{1+\rho_3}{1-\rho_2} - idle_1\frac{1-\rho_3}{1+\rho_1}) - (rtt_2 - D_2 - idle_2\frac{1-\rho_3}{1+\rho_2}),$$
$$\left. r_3 - (s_1 - r_1)\frac{1-\rho_3}{1+\rho_1} - (s_2 - r_2 + D_1)\frac{1-\rho_3}{1+\rho_2} - D_2\right] \tag{4.23}$$

Node N

$$\left[r_N - (1+\rho_N)\sum_{i=1}^{N-1}\frac{s_i - r_i + rtt_{i-1} - D_{i-1}}{1-\rho_i} - (rtt_{N-1} - D_{N-1})\right.$$

$$+(1-\rho_N)\sum_{i=1}^{N-1}\frac{idle_i}{1+\rho_i},$$

$$r_N-(1-\rho_N)\sum_{i=1}^{N-1}\frac{s_i-r_i+D_{i-1}}{1+\rho_i}-D_{N-1}\Bigg] \tag{4.24}$$

The interval for node 1 consists of the single point $h_1(t_E)$. For node 2 the amount of time $s_1 - r_1$ (during which the message was stored in node 1 after being generated and before being sent) is subtracted from the message arrival time r_2. The difference $rtt_1 - idle_1$ between round trip and idle time is used as an upper bound for the message delay, the minimum delay D_1 is used as a lower bound. Transforming time intervals between the different time scales as described in Section 4.4.3 results in the interval shown for node 2. Continuing this way with subtracting total node storage time from message arrival time and using the sum of round trip minus idle times as the upper bound for message delay, and assuming $rtt_0 = 0$ and $D_0 = 0$, one will end up with the interval shown for node N.

4.4.6 Interval Arithmetic

Using the algorithm described in the previous sections, we are now able to decide temporal predicates over time stamps using a variant of the interval arithmetic described in [2]. To decide whether $[h_1^l, h_1^r]$ happened before $[h_2^l, h_2^r]$, for example, the following rule can be used:

$$[h_1^l, h_1^r] < [h_2^l, h_2^r] = \begin{cases} \text{YES} & : \quad h_1^r < h_2^l \\ \text{NO} & : \quad h_2^r < h_1^l \\ \text{MAYBE} & : \quad \text{otherwise} \end{cases} \tag{4.25}$$

To determine whether $[h_1^l, h_1^r]$ and $[h_2^l, h_2^r]$ happened within a certain real-time interval T, the following rule is used:

$$\left|[h_1^l, h_1^r] - [h_2^l, h_2^r]\right| < T =$$

$$\begin{cases} \text{YES} & : \quad \max(h_2^r, h_1^r) - \min(h_2^l, h_1^l) < T(1-\rho_{\max}) \\ \text{NO} & : \quad \max(h_2^l, h_1^l) - \min(h_2^r, h_1^r) \geq T(1+\rho_{\max}) \\ \text{MAYBE} & : \quad \text{otherwise} \end{cases} \tag{4.26}$$

Note that the real-time interval T has to be transformed to local time first by multiplying with $1 \pm \rho_{\max}$, since all time-stamp intervals refer to the time scale defined by the local hardware clock.

The real-time "distance" between two time-stamp intervals can be estimated using the following formula:

$$\left|[h_1^l, h_1^r] - [h_2^l, h_2^r]\right| \leq (\max(h_2^r, h_1^r) - \min(h_2^l, h_1^l))/(1-\rho_{\max}) \tag{4.27}$$

Again, the calculated local time difference has to be transformed to real-time by dividing by $1 - \rho_{\max}$.

When comparing points in time (for example a locally generated $h(t_x)$) with time-stamp intervals received from other nodes, $h(t_x)$ can be treated as a time-stamp interval $[h(t_x), h(t_x)]$ and used with the above equations.

4.4.7 Implementation

The basic idea for implementing the algorithm is to incrementally calculate the three sums in the Formula 4.24 along the message path. The implementation assumes an asynchronous, reliable communication mechanism but can easily be extended to unreliable communication mechanisms (e.g., by means of timeouts and retransmissions).

A time stamp can be represented in the following way using C:

```
struct TimeStamp {
    Time begin, end, received;
    Time s1, s2, s3;
};
```

where `begin` and `end` are the left and right ends of the time-stamp interval, `received` is the time of arrival, and where `s1`, `s2`, `s3` are the three sums in Formula 4.24 from left to right, which are incrementally calculated as the message is forwarded from node to node. Note that `begin`, `end`, and `received` are local variables that don't need to be transmitted between nodes. One or more instances of `TimeStamp` can be contained in an application message.

`Time` is a representation for points in time and time differences. Computer clocks are discrete, so an integer type would be appropriate. But care has to be taken because of time transformations, which may result in fractional values, so either a floating-point type must be used or the results have to be rounded such that the integer interval always contains the floating-point interval. Here we assume floating point values.

The generator of a time-stamped message performs the following actions:

```
1 Generator:
2   TimeStamp S;
3   S.begin = S.end = S.received = NOW;
4   S.s1 = S.s2 = s3 = 0;
```

where `NOW` refers to the current value of the local clock. As explained in the previous section, the interval is initialized to current time in the node. All other fields are set to zero.

A time-stamped message is sent using the following actions:

```
 1 Sender:
 2   TimeStamp S; /* locally generated or received */
 3   Time idleend = NOW;
 4
 5   IF (idlebegin[receiver] == 0 OR
 6       idleend - idlebegin[receiver] > max_idle)
 7   THEN
 8       send <sync> to receiver;
 9       receive <ack> from receiver;
10       idleend = NOW;
11       idlebegin[receiver] = idleend;
12   ENDIF
13
14   send <xmit(S, idleend - S.received,
15         idleend - idlebegin[receiver],
16         local_rho)> to receiver;
17   receive <ack(resend)> from receiver;
18   idlebegin[receiver] = NOW;
19
20   IF (resend == TRUE) THEN
21       idleend = NOW;
22       send <xmit(S, idleend - S.received,
23             idleend - idlebegin[receiver],
24             local_rho)> to receiver;
25       receive <ack> from receiver;
26       idlebegin[receiver] = NOW;
27   ENDIF
```

The sender first checks if the time when the last message was sent to the receiver (line 5) is unknown or if the idle time is too large (line 6). If suitable values for *rtt* and *idle* are not available, a `sync` message is sent before waiting for an `ack` to initialize `idlebegin[receiver]`. Then the sender transmits the `TimeStamp` data structure to the destination node along with the amount of time the message was stored in the current node (line 22) and the idle time (line 23) according to the local time scale with maximum clock drift `local_rho`. Then an acknowledgment containing a parameter `resend` is awaited. If `resend` is true, then the message is sent again in order to enable the receiver to measure round-trip time.

The receiver of a message performs the following actions:

```
1 Receiver:
2   IF (receive <sync> from sender) THEN
3       rttbegin[sender] = NOW;
4       send <ack> to sender;
5   ELSEIF (receive <xmit(S, lifetime, idletime,
6                    rho)> from sender)
```

```
 7   THEN
 8       Time rttend = NOW;
 9       IF (rttbegin[sender] == 0) THEN
10           rttbegin[sender] = NOW;
11           send <ack(TRUE)> to sender;
12       ELSE
13           S.s1 += lifetime/(1 - rho);
14           S.s2 += idletime/(1 + rho);
15           S.s3 += lifetime/(1 + rho);
16           S.begin = rttend
17              - S.s1*(1 + local_rho)
18              + S.s2*(1 - local_rho)
19              - (rttend - rttbegin[sender]) + D;
20           S.end = rttend
21              - S.s3*(1 - local_rho)
22              - D;
23           M.s1 += (rttend - rttbegin[sender] - D)
24              * (1 - local_rho);
25           M.s3 += D*(1 + local_rho);
26           rttbegin[sender] = NOW;
27           send <ack(FALSE)> to sender;
28       ENDIF
29   ENDIF
```

The receiver waits for a `sync` or `xmit` message from a sender. If it receives a `sync`, it just initializes `rttbegin[sender]` and returns an `ack` to the sender.

If an `xmit` message is received, then the receiver first checks if the time of arrival of a previous message from this sender is known (line 9). If not so, then `rttbegin[sender]` is initialized and an `ack(TRUE)` message is returned to the sender, asking for a retransmission.

If `rttbegin[sender]` is known in the sender, then the fields of the received time stamp `S` are updated according to Equation 4.24 and `S.begin` and `S.end` are calculated. Note that the received value for `S.s1` (`S.s3`) does not yet include the last rtt and D values, so it has to be explicitly subtracted in lines 19 and 22 without time transformation, since rtt and D have been measured with respect to receiver's time scale. Only after `S.begin` and `S.end` have been calculated, `M.s1` (`M.s3`) are updated accordingly. Finally, an `ack` is sent back to the sender.

The checks for `idlebegin[receiver]` and `rttbegin[sender]` in sender and receiver, respectively, enable both the sender and receiver to independently drop entries from the sets `idlebegin` and `rttbegin` in order to limit the memory footprint as described in Section 4.4.4.

4.4.8 Evaluation

In this section we examine the compliance of TSS with the region in the design space indicated in Section 4.3.

Intermittent Connectivity

If a message can be forwarded from a source node to a destination node (possibly across temporary partition boundaries), synchronization can be performed as well, provided that connections between adjacent nodes remain established long enough for at most two round-trip message exchanges.

Resource Efficiency

In most cases, TSS can piggyback on existing network traffic, resulting in a slight increase of the size of messages that carry a time stamp. With the implementation in Section 4.4.7, every message would contain the partial sums `s1`, `s2`, `s3`, `lifetime`, `idletime`, and `local_rho`. Assuming `sizeof(Time) = 4` and `sizeof(local_rho) = 1`, the size of a message would increase by 17 bytes compared to a simple time-stamp of type `Time`. Note that `local_rho` could be cached by the receiver. It might also be worthwhile to use a variable bit-length encoding for the various `Time` values, since `lifetime` and `idletime` (but also the partial sums) tend to be small in unpartitioned networks.

Additional messages are only needed when a pair of nodes first communicates or when two nodes haven't communicated for a long time. Hence, additional overhead is only introduced when a pair of nodes has been "idle" before. Under "heavy load", when a pair of nodes communicates frequently, no additional messages are required.

Correctness

Since time-stamps are represented as intervals in TSS, temporal predicates can always be decided correctly or not at all according to the rules in Section 4.4.6.

Precision

In order to get an impression of the precision of the algorithm TSS we performed some measurements on a cluster of 800 MHz Pentium III Linux PCs connected by 100 Mbit/s Ethernet using TCP and assuming $\rho = 10^{-6}$. This has to be considered as a best case scenario, since sensor networks typically use a networking technology providing a bandwidth well below 1 Mbit/s and embedded processors with no more than 10 MIPS. However, since the algorithm is neither especially CPU intensive nor network-bandwidth intensive, the measurements should give a good impression of the algorithm's possible precision.

Synchronization inaccuracies show up as time-stamp intervals of increasing length and stem from two different sources. Firstly, due to the clock drift, interval length increases with the age of a time stamp. Secondly, the interval length

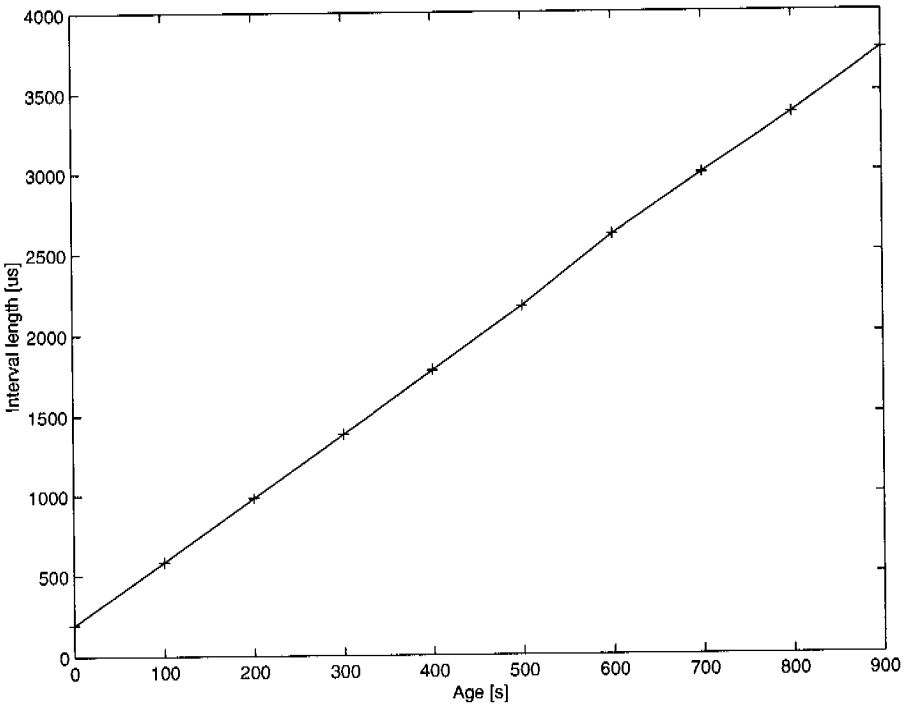


Figure 4.8: Precision depending on age.

increases with the number of hops a time stamp has been passed along, due to the estimation of messages delays.

We performed two measurements, the first of which examines time-stamp interval length as a function of time-stamp age. Since the error resulting from age is additive over the nodes, we generate a zero-length time stamp interval in node 1, store it for X seconds before forwarding it to node 2, which prints out the length of the received time-stamp interval. We repeat this experiment 1000 times and calculate averages. Figure 4.8 shows the results[1], indicating a linear increase of imprecision with age.

The second measurement examines time-stamp interval length as a function of the number of hops a time stamp has been passed along. We generate a zero-length time-stamp interval in node 1 and pass it on to node 2, 3, ..., 7, which all print out the length of the received time-stamp interval. We repeat this experiment 1000 times and calculate averages. Figure 4.9 shows the results[2], indicating a linear increase of inaccuracy with the number of hops.

Since the two types of inaccuracies are additive, one can interpret the measurements as follows: Passing a time stamp along no more than 5 hops with an age of no more than 500 seconds, one can expect an inaccuracy of no more than 3ms in the examined setting. That is, exact results (as opposed to MAYBE) can be obtained as long as compared time stamps represent points in time which are more

[1]The exact interval lengths are 195, 585, 982, 1378, 1775, 2170, 2609, 2992, 3369, 3764μs.

[2]The exact interval lengths are 0, 201, 400, 562, 752, 926, 1113, 1273μs.

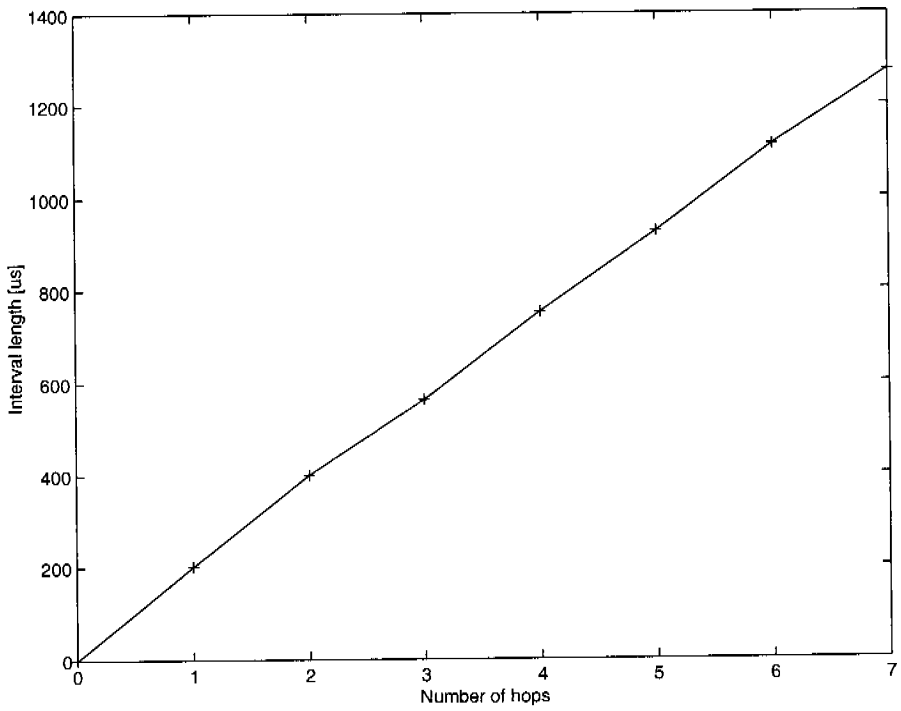


Figure 4.9: Precision depending on number of hops.

than 6ms apart, since then the time-stamp intervals (3ms each) cannot overlap. With less than 6ms difference, an exact answer may still be obtained, but MAYBE results are likely.

These measurements indicate that "collocated events" (both in time and space) can be synchronized with good precision. The larger the distance between the events in time and space, the lower is the precision.

4.4.9 Potential Improvements

There are several potential approaches to improve the accuracy of the algorithm (i.e., reduce the probability of MAYBE results), which might be worth further investigation. In general, these techniques tend to improve precision by introducing additional overheads.

One idea to avoid MAYBE results when comparing time stamps originating from the same node is to keep a history of time stamps instead of only one time stamp. Instead of *updating* a time stamp upon receipt, the receiving node *appends* the updated time stamp together with a unique node identifier i and ρ_i to the time stamp history or *reuses* a time stamp from the history if there is an entry for node i in the history already. If comparing time stamps results in MAYBE, then the histories of the compared stamps are searched for common nodes and the comparison is repeated using the time stamps of these common nodes, transforming time values if necessary, and using the according ρ values from the histories. This is likely to give a "better" (i.e., non-MAYBE) answer, since imprecision increases

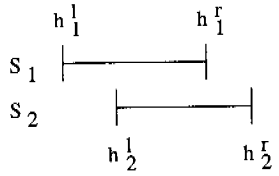


Figure 4.10: Overlapping time stamp intervals.

with age and hop count of the time stamps. For the same reason the accuracy of calculated real-time spans can be improved by using "younger" time stamps from the history in the same way whenever possible.

A different and more general approach is to replace MAYBE results with a probability depending on the relative arrangement of the compared time-stamp intervals. The algorithm would then output "$X < Y$ with probability p" instead of "$X maybe < Y$". To implement this, we have to derive probability distributions for the exact time instants over the time-stamp intervals.

Consider for example the two overlapping time-stamp intervals S_1 and S_2 with $h_1^l \leq h_2^l \leq h_1^r \leq h_2^r$ shown in Figure 4.10, for which the algorithm would output "$S_1 maybe < S_2$". If we knew probability distributions $p_1(h)$ and $p_2(h)$, such that $p_i(h_i)$ is the probability that the exact point in time represented by S_i is h_i, we could calculate the probability p for $S_1 < S_2$ by "iterating" over the possible h_1 values and summing up the probabilities for $h_1 < h_2$:

$$\int_{h_1^l}^{h_1^r} p_1(h_1) \left(\int_t^{t_2^r} p_2(h_2) \mathrm{d}h_2 \right) \mathrm{d}h_1 \tag{4.28}$$

For uniform distributions $p_i(h)$[3] this evaluates to

$$\frac{h_2^l - h_1^l}{h_1^r - h_1^l} + \frac{h_2^r(h_1^r - h_2^l) - ({h_1^r}^2 - {h_2^l}^2)/2}{(h_1^r - h_1^l)(h_2^r - h_2^l)} \tag{4.29}$$

Assuming, for example, $h_1^l = 0$, $h_1^r = h_2^r = 2$, and $h_2^l = 1$, we can calculate the probability for $S_1 < S_2$ as 0.75. Assuming a uniform distribution, however, usually is an oversimplification, since due to the characteristics of the algorithm[4] the probability in the middle of the interval is much higher than at the ends. It remains an open task to obtain good probability distributions. Furthermore it has to be investigated for which cases knowing a probability instead of MAYBE is advantageous for applications.

[3] $p_1(h) = 1/(h_1^r - h_1^l)$ for $h \in [h_1^l, h_1^r]$ and 0 otherwise; $p_2(t)$ likewise.

[4] We use D and rtt as lower and upper bounds for the message delay. It is much more likely that the actual message delay is about $rtt/2$ than D or rtt.

4.5 Summary

This chapter is devoted to time synchronization in wireless sensor networks. We first discussed important models with respect to hardware clocks and communication. Following the framework developed in Chapter 3, we then presented important concepts used by time synchronization algorithms. Referring to these concepts, we then presented and discussed important existing approaches to time synchronization – both in traditional distributed systems and for sensor networks. We then identified a region in the design space that cannot be appropriately supported by these existing algorithms. In particular, we showed that networks with intermittent connectivity are not sufficiently supported. We presented and evaluated Time-Stamp Synchronization to support this region in the design space. A key feature of Time-Stamp Synchronization is that rather than synchronizing clocks, time stamps are transformed between the unsynchronized time-scales of sensor nodes. We showed that this approach is efficient since time information can be piggybacked on existing message exchanges. Based on a prototypical implementation, we evaluated the precision of our approach and found that imprecision grows linearly with the age of time stamps and with the hop-distance between nodes, providing an accuracy in the order of milliseconds. Finally, we discussed potential improvements in settings where precision is more important than efficiency.

Chapter 5

Sensor Node Localization

The significance of physical space for sensor networks has been reflected by the development of a number of node localization algorithms in the recent past. However, most of these approaches have been designed for "traditional" sensor networks, covering only a small region of the design space discussed in Section 2.2. We will identify an important region in the design space that is not sufficiently supported by existing algorithms. In order to fill this gap, we present and evaluate an approach called "Lighthouse Location System".

We begin our discussion in Section 5.1 by studying fundamental system models and by presenting concrete algorithmic techniques for localization. The discussion will be structured according to the common framework we developed in Chapter 3. Referring to these techniques, we present existing algorithms for node localization in Section 5.2. Our algorithm will be motivated and presented in Sections 5.3 and 5.4.

5.1 Background

This section reviews models and concepts for node localization. The discussion is structured according to Section 3.3.

5.1.1 Signal Propagation and Mobility Models

In the following two subsections we discuss models of signal propagation and of node mobility. The propagation of signals such as radio or sound are fundamental for measuring spatial relationships among nodes. Models of node mobility can help maintain localization over time.

Signal Propagation Models

Many localization approaches for sensor networks assume that sensor nodes are equipped with hardware for measuring spatial relationships among nodes such as

distance or direction. For this purpose, some systems reuse existing radios originally intended for data communication, others introduce additional sensors and actuators to achieve a satisfactory precision. Besides radio, audio is perhaps the most commonly used signal modality for this purpose.

Two signal propagation features commonly used for measuring spatial relationships are *propagation delay* and *received signal strength*, since both are functions of the distance between emitter and receiver of a signal. Deriving spatial relationships from a measured propagation delay requires a precise model of the propagation speed of the signal. The derivation of spatial relationships from received signal strength requires a precise model of signal attenuation.

Signal propagation speed is mainly a function of the signal modality, its frequency, and of the medium. For example, the propagation speed of radio waves can diverge significantly from c in vacuum and air. In water (and wet materials), for example, propagation speed is about $c/1.33$. The radio propagation speed is also influenced by ionizing radiation (e.g., solar winds), where the speed reduction depends on the frequency of the signal. With acoustic waves, the propagation speed varies between about 60m/s (e.g., rubber) and about 6000m/s (e.g., stone). In air, propagation speed depends on factors such as temperature, and – to a lesser extent – humidity and pressure. For example, propagation speed in air at -20°C is 320m/s, whereas 344m/s are found for +20°C. A more precise model for speed of sound in air is:

$$c_{\text{air}} = \sqrt{\gamma RT} \tag{5.1}$$

where T is absolute temperature, R is the gas constant (286 $m^2/s^2/K$), and where γ is the heat capacity ratio (about 1.4 for typical humidity and pressure values). The effective speed of sound also depends on movements of the medium such as winds and convection streams.

Typical localization systems use one out of the following three approaches to model propagation speed. Firstly, a constant speed is assumed that may be calibrated under operating conditions using separate calibration equipment. Secondly, additional sensors may be attached to nodes in order to measure parameters that influence the propagation speed (e.g., temperature for speed of sound). Thirdly, the propagation speed is treated as an unknown variable in addition to the sought location. Additional constraints are required in order to solve for propagation speed *and* location.

The received signal strength depends on the emitted signal strength and the attenuation. Attenuation is a function of the signal modality, frequency, and the medium. In general, an omni-directional source radiating into free space can be modeled as

$$P = kP_s/r^e \tag{5.2}$$

where $e = 2$, P_s is the emitted power, r is the distance between sender and receiver, and k is a constant factor. The quadratic relationship stems from the fact that the emitted power is equally spread over the points on a sphere with radius r.

In physical structures where signal propagation is limited to certain regions (e.g., corridors), smaller attenuation exponents between 1.5 and 2 may be observed.

Additional sources of attenuation are absorption of the medium, interaction of the signal with reflective surfaces, and interaction of the signal with obstacles between emitter and receiver. Radio waves, for example, are absorbed by ionized gases. Sound waves are absorbed in gases due to friction between molecules that results in heat generation. Reflective surfaces may lead to the situation that a reflected signal interferes with the unreflected (or another reflected) signal at the receiver. This can be observed, for example, if emitter and receiver are near the ground or if the environment is cluttered with reflective surfaces. Radio waves often experience a phase shift close to 180° when reflected, which often leads to destructive interference. Hence, for near-ground radio communication the exponent e in Equation 5.2 is close to 4. In highly cluttered environments with many reflections interfering at the receiver, e may be as large as 6. Sound waves are less susceptible to such interference effects, since the above mentioned phase shift is typically not found. For near-ground communication, for example, e is still close to 2.

Additional sources of attenuation for sound in air are meteorological conditions such as wind-velocity gradients (e.g., due to friction with the ground) and temperature gradients (e.g., due to heat disseminated by the ground). Both result in gradients in sound velocity which lead to a refraction effect, such that sound waves are bent upwards or downwards with respect to the ground, depending on the relative direction of the waves to the gradient. One further source of attenuation of sound in air are random fluctuations of wind and temperature which cause fluctuations in amplitude and phase of the signal at the receiver.

The above discussion should make it obvious that models for signal attenuation do heavily depend on the environment where they are used in. Note that even small changes in the locations of emitter, receiver, or reflective surfaces may cause significant changes in received signal strength due to changed conditions for multi-path interference. This is particularly relevant for radio waves, where multi-path interference can have a significant impact on the attenuation exponent in Equation 5.2. Hence, large errors must be expected if the model does not correctly represent reality. Typical models used in localization systems are based on Equation 5.2 with $e = 4$. Range measurement errors based on such models can be in the order of 40% and above [88].

The use of signal propagation features for range measurements obviously requires that the signal can travel along the line-of-sight path between the emitter and the receiver. If the direct line is blocked by non-transparent obstacles, the signal may reach the receiver due to reflections. Since the effective length of such reflected paths is longer than the line of sight, the resulting distance estimates will be greater than the Euclidean distance.

Many of the above signal propagation properties depend on the actual frequency of the signal. Hence, the use of wide-band signals (which contain many different frequencies) can help mitigate or compensate errors that are due to frequency-dependent propagation properties. Also, wide-band techniques are robust to in-

terference with narrow-band signals emitted by other sources in range (e.g., microwave ovens, acoustic background noise) [42, 43]. Likewise, the use of different signal modalities (e.g., sound and radio) can help improve robustness and precision (cf. Section 2.7.2).

Mobility Models

Mobility models are mechanism to describe possible movements and locations of mobile entities. They can be a valuable tool for improving the precision of localization approaches since they provide hints on the possible locations of a sensor node. Two very simple mobility models are the "static model" where nodes do not move at all and the "unconstrained mobility model" where nodes can be at any place anytime. Between these two extremal models, a number of more realistic models can be found.

Basically, a mobility model imposes constraints on the possible location, speed, and acceleration of a mobile entity. These constraints can be either static or dynamic. In static models, the same constraints apply always. In dynamic models, the constraints may be a function of time or of previous system states. If, for example, a sensor node is attached to a car, possible node locations are constrained to roads (with high probability), there are bounds on the speed and acceleration of the sensor node. While these constraints are static, we can also identify dynamic constraints, where future locations of the sensor node are constrained by its previous locations. For example, a car is very unlikely to leave a crossing via the same road it entered. Given such a set of constraints, the location of the node at some time t_0 (and possibly also earlier locations of the node), we can derive a set of possible locations of the node at some time $t_1 > t_0$ without making any measurements.

A number of mobility models are commonly used to study the behavior of mobile systems, for example the random waypoint model (a node chooses a random destination point and moves there on a straight line with constant speed, stops for some time before choosing a new destination). Another common class of mobility models are graph-based, where a node can only move along the edges of an Euclidean graph. At each vertex, one of the outgoing edges is chosen according to specific rules (e.g., random walks).

5.1.2 Obtaining Constraints

There are two basic types of spatial relationships that are used as constraints for localization: distances and angles.

Distance Constraints

Distances are used in two variants: absolute distance to a reference node and distance differences to two or more reference nodes. Absolute distances can be obtained by measuring time-of-flight or received signal strength, using propagation

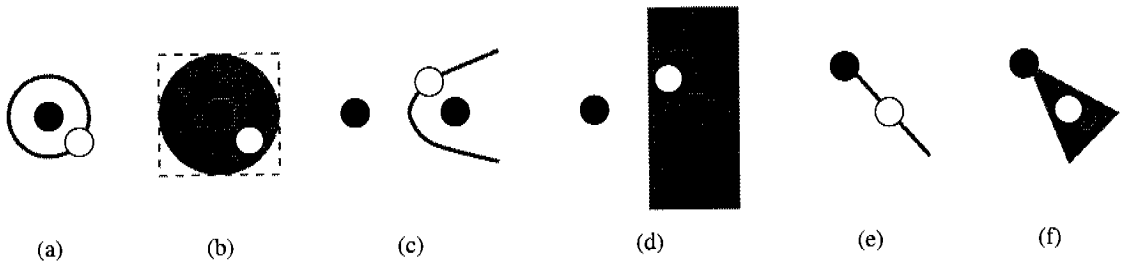


Figure 5.1: References (black) impose constraints (red regions) on the locations of a client node (white). (a) Distance estimate, (b) distance bound, (c) distance difference, (d) closest reference, (e) direction of arrival estimate, and (f) direction of arrival bounds.

models as described in Section 5.1.1. Note that time-of-flight measurements require some form of time synchronization between sender and receiver. The precision of synchronized time is determined by the precision that is required for the distance estimate. For received signal strength, the emitted signal strength must also be known. The resulting distance constraint is illustrated in Figure 5.1 (a).

While the above approach typically results in point estimates for distance, there is also a notable method for obtaining bounds based on network connectivity. If the communication range of a sensor node is at most R, then two nodes that can hear each other are at most R apart. The resulting constraint is illustrated in Figure 5.1 (b). In some cases it is preferable to use a bounding box instead of a distance constraint as indicated by the dashed rectangle in (b). For example, the intersection of two bounding boxes is also a bounding box, while the intersection of two spheres is not a sphere anymore.

Distances and distance bounds can also be combined across multiple hops to obtain distance bounds for client nodes that are not network neighbors of a reference node. If two nodes are separated by N hops, their distance is at most NR. Similarly, if the distances along a path between two nodes are r_1, ..., r_N, than the distance between the nodes is at most $\sum r_i$.

Distance differences can be obtained by measuring the time difference(s) between the arrival of signals emitted by two or more synchronized reference nodes, or by measuring the differences between the received strengths of signals emitted by two or more reference nodes with identical emitted signal strengths. Distance-difference estimates can then be obtained by applying the propagation models to the time or signal strength differences. Note that the resulting constraint then involves multiple reference nodes as exemplified in Figure 5.1 (c). Figure (d) illustrates a bound-based variant of distance differences, where a node determines to which of a set of references it is closest (i.e., reference with strongest received signal or reference with earliest time of arrival).

Angular Constraints

Angles are mainly used in form of direction of arrival. If a node is located at the origin of a 2D coordinate system, then the direction of arrival can be modeled as the angle enclosed between the x axis and a line connecting origin with the reference node. In order to measure direction of arrival, complex hardware is typically required. One possible approach are sector antennas, where multiple directional antennas are arranged such that each possible sender can be received by at least one antenna. Since each antenna has a well-defined angle of beam spread, bounds on the angle of incidence can be obtained. As illustrated in Figure 5.1 (f), this results in a constraint on the location of the node. Another approach is the use of multiple omnidirectional receivers with known baselines in between. By measuring the time difference of arrival at these receivers, the direction of arrival can be estimated, which results in a constraint on the node location as illustrated in Figure 5.1 (e).

Note that it is also possible to combine angular and distance measurements. However, due to the hardware overhead, increased size and cost of sensor nodes, angle-based approaches are barely used for sensor networks.

5.1.3 Combining Constraints

A single constraint can be interpreted as a set of possible node locations that typically form a connected region in space as illustrated in Figure 5.1. Combining multiple constraints is then equivalent to computing the intersection of the respective regions. In some cases the exact shape of the intersection region is required, sometimes a single point that lies in the intersected region is computed, and in some cases a simple shape (e.g., bounding box or sphere) that is enclosed by or enclosed the intersection region is sought.

Typically an intersection is derived analytically, but there are situations where this is infeasible or undesirable due to computational overheads. A notable alternative is to partition space into pixels or voxels (i.e., 3D pixels), such that the edge length of a pixel equals the required precision. A single bit is associated with each pixel, which is initially set to "1". For each available constraint, the pixels that are outside the region represented by the constraint are set to "0". Eventually, the remaining "1" pixels form an approximation of the intersection region.

Due to measurement errors it may occur that a given set of constraints has an empty intersection. There are basically two approaches to deal with such cases. Firstly, outliers can be rejected, such that a set of constraints with a non-empty intersection is obtained. As for time synchronization, various criteria can be applied to control selection of these constrains (e.g., minimize number of rejected constraints, minimize intersection region). Secondly, a solution (typically a single point) can be computed that minimizes a certain error metric. For example, the error of a point solution with respect to a single constraint can be defined as the smallest distance between the solution point and any point contained in the region defined by the constraint. The sought solution should then minimize the sum of

the errors for all constraints. If confidence values for the constraints are available, the terms of this sum could also be weighted according to the confidence values.

Multilateration

One of the most commonly used approaches is multilateration, where multiple constraints of the type depicted in Figure 5.1 (a) are combined and a point solution (x, y, z) is sought. Each constraint is then of the form (x_i, y_i, z_i, r_i), where (x_i, y_i, z_i) is the location of a reference node, and where r_i is the distance of the client node from this reference. In general, four constraints are required to define a unique solution for (x, y, z). If there are no measurement errors, the solution can be obtained by solving the following non-linear equation system for $1 \leq i \leq 4$:

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = r_i^2 \tag{5.3}$$

This non-linear equation system can be transformed into a linear equation system by subtracting the equation for $i = 1$ from the remaining three equations. This will eliminate the quadratic terms x^2, y^2, z^2 and yields the following linear equation system for $2 \leq i \leq 4$:

$$2x(x_1 - x_i) + 2y(y_1 - y_i) + 2z(z_1 - z_i) = r_i^2 - r_1^2 + (x_1^2 + y_1^2 + z_1^2) - (x_i^2 + y_i^2 + z_i^2) \tag{5.4}$$

Standard methods can be used to solve for (x, y, z). If more than four constraints are available, a similar, but over-constrained linear equation system of the form $A \cdot (x, y, z)^T = b$ must be solved. A standard error metric for finding an approximate solution is to minimize the sum of the squared errors $\sum_i (A_i \cdot (x, y, z) - b_i)^2$. The according solution can be found by solving the linear equation system

$$A^T A \begin{pmatrix} x \\ y \\ z \end{pmatrix} = A^T b. \tag{5.5}$$

Above we noted that time-of-flight measurements require time synchronization between sender and receiver. This can be either achieved by explicit time synchronization (cf. Chapter 4), or by including the time offset Δt between sender and receiver as an additional variable in the equation system 5.3, yielding

$$(x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 = (r_i - \Delta t c)^2 \tag{5.6}$$

where c is the (average) propagation speed of the signal. As with Equation system 5.3, a linear equation system can be obtained by subtracting the first equation. However, at least one additional constraint is needed in order to obtain a unique solution for $(x, y, z, \Delta t)$. Note that the same approach can be used if Δt is known, but the propagation speed c of the signal is unknown (e.g., speed of sound in air).

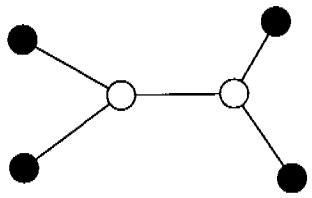


Figure 5.2: Four references with known locations define unique locations for two nodes if distances between connected nodes are known. Traditional multilateration cannot be applied, since there are too few constraints for each individual node.

Collaborative Multilateration

In some node constellations too few constraints may be available to obtain a unique solution for (x, y, z). However, by including other nodes with unknown locations into consideration, a unique solution can often be found anyway. As depicted in Figure 5.2, four black reference nodes define unique locations for two client nodes if the distances between connected nodes are known. Obtaining a solution for the locations of two or more client nodes in such situations is referred to as collaborative multilateration. One possible approach to find a solution for such problems is to request that a measured distance r_{ij} between nodes i and j should be matched by the distance $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}$ between the sought location estimates (x_i, y_i, z_i) and (x_j, y_j, z_j) for nodes i and j. An optimization problem can be formulated by minimizing the sum of the squared errors over all edges (i, j):

$$\min \sum_{(i,j)} \left(r_{ij} - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \right)^2 \tag{5.7}$$

The resulting non-linear optimization problem typically does not have a closed-form solution and numerical approximations must be applied to find the solution.

Centroids

This approach considers the problem of finding a point in the intersection of a number of distance-bound constraints (cf. Figure 5.1 (b)) for a single node. Such a point can be obtained by computing the average (i.e., centroid) of the locations of all references that define constraints on the node.

Triangle Test

This approach uses distance-difference constraints (cf. Figure 5.1 (d)) to decide whether a node is contained in a triangle formed by three reference nodes with known locations. This test is based on the following property: a node located outside the triangle can be moved, such that the distances to all references are either all increased or all decreased simultaneously. In contrast, all movements of a node located inside the triangle will increase the distance to some references and decrease the distance to other references simultaneously.

Based on this observation, the following approximative triangle test can be used in dense networks: a node is assumed to be in the triangle, if no neighbor of the node is further from or closer to all three references simultaneously. This test can be performed for a large set of different triangles, resulting in a set of "inside" triangles for each node. By intersecting these triangles, bounds on the node location can be obtained. A point estimate can be computed as the center of gravity of the intersection region.

5.1.4 Maintaining Localization

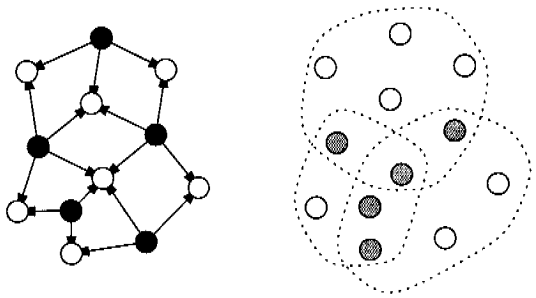
If nodes are mobile, the precision of an instantaneous location estimate will degrade over time. The conceptually simplest approach to maintain up-to-date location estimates is to re-execute the localization algorithm frequently. There are various ways to trigger such a re-execution. Firstly, the algorithm can be re-executed after a fixed amount of time. Secondly, if a certain imprecision can be tolerated, the maximum amount of time between executions can be calculated using a mobility model. Thirdly, execution can be triggered whenever the application requests an estimate of the current location. Fourthly, execution can be triggered whenever a node moves, which can be detected with simple motion detection sensors or accelerometers that are attached to sensor nodes. Which of these approaches is most appropriate depends on various system parameters such as degree of node mobility, tolerated imprecision, and the frequency with which a node requests a location estimate. However, as mentioned in Section 3.3.5, the applicability of repeating execution is limited by the time needed by the execution of the algorithm, and by available resources.

Dead Reckoning

Another approach to maintain localization is the use of dead reckoning techniques, where a node is equipped with sensors to measure movements. By integrating these movements with an earlier location estimate, an estimate of the current location can be obtained. Common approaches for dead reckoning are the analysis of wheel rotations (e.g., vehicles and robots) and the use of accelerometers. If position $x(t_0)$ and velocity $v(t_0)$ at some time t_0 are known, then an estimate of $x(t_1)$ can be obtained by measuring acceleration $a(t)$:

$$x(t_1) = x(t_0) + \int_{t_0}^{t_1} \left(v(t_0) + \int_{t_0}^{t} a(t')\mathrm{d}t' \right) \mathrm{d}t \tag{5.8}$$

By using three accelerometers with mutual perpendicular axes, this approach can be extended to three dimensions. However, due to the quadratic relationship between $a(t)$ and $x(t)$, small measurement errors in $a(t)$ will accumulate over time and show up squared in $x(t)$. Hence, the precision of such an approach decreases quickly with growing $t_1 - t_0$.



(a) (b)

Figure 5.3: Overlay topologies used by localization algorithms. (a) Stars, (b) clusters.

Prediction

Prediction based on mobility models is another approach to maintain localization over time. Here, past locations of a node at time $t \leq t_0$ are used with a mobility model to obtain constraints on the location of the node at time $t_1 > t_0$. With a mobility model where velocity is limited by $0 \leq v \leq v_{\max}$, for example, the following constraint can be derived:

$$x(t_0) \leq x(t_1) \leq x(t_0) + v_{\max}(t_1 - t_0) \tag{5.9}$$

Note that depending on the used mobility model, the obtained constraints may become very loose with growing $t_1 - t_0$. Overall, both dead reckoning and prediction are helpful approaches to bridge short gaps between consecutive executions of a localization algorithm. While prediction and dead reckoning are very commonly used for time synchronization (cf. Section 4.1.4), they are currently barely used for localization as far as the domain of sensor networks is concerned.

5.1.5 Selecting Constraints

Most localization algorithms are unstructured, that is, they use any available nodes in the neighborhood for obtaining constraints. The reason for this is that in order to achieve a good precision, measurement errors must often be compensated by a large number of constraints.

Two commonly used overlay topologies are depicted in Figure 5.3. With the star topology depicted in (a), nodes use all available references in their neighborhood to estimate their location, but nodes with estimated locations are not used as references for other nodes. In (b), nodes in a cluster establish a local coordinate system and estimate their locations with respect to this reference grid. Adjacent clusters share a number of nodes to allow for the derivation of a coordinate transformation between these clusters.

5.2 Related Work

This section presents and discusses existing approaches for node localization. We first discuss related and influential approaches from other domains, pointing out their shortcomings in the context of sensor networks. Our main focus is, however, on algorithms which have been specifically developed for sensor networks. These are based on the models and concepts presented in the previous section.

5.2.1 Traditional Localization Approaches

In this section we consider two representative localization systems that have been developed for applications other than sensor networks.

Global Positioning System

GPS [47] is based on 24 satellites orbiting the earth. All satellites broadcast signals that are used for time-of-flight measurements and which carry additional information such as current time and location of the satellites. The satellites are very accurately synchronized with each other. A GPS receiver uses a so-called almanac and the information transmitted by the satellites to maintain precise estimates of the current locations and time of the satellites. The almanac is essentially a (mobility) model of the satellite orbits.

In order to obtain a location estimate, a GPS receiver measures time-of-flight to at least four satellites and uses a variant of Equation system 5.6 to obtain an estimate of its location and time offset. Only four (instead of five) satellites are needed since the receiver is expected to be below the satellites.

A precision of (100m horizontal, 156m vertical, 340 nanoseconds) can be achieved for civilian users and (22m horizontal, 28m vertical, 200 nanoseconds) for military users 95% of the time. This precision can be further improved by differential GPS, where a ground station with known location measures its location with GPS in order to compute correction signals for all visible satellites. These correction signals are broadcast and used by receivers to correct their measurements. Differential GPS can provide a precision of about 1-10m.

Despite its global availability, GPS is of rather limited value for sensor networks mainly due to resource constraints (GPS receivers are expensive and consume a significant amount of power) and due to the required free line of sight to at least four satellites. Some researchers propose to equip anchor nodes with GPS.

Active Bats

Active Bats [104] are in indoor location system that consists of mobile tags ("bats") that can be located and a wired infrastructure. Bats are capable of receiving radio messages and of transmitting ultrasonic signals. The infrastructure mainly consists of ultrasound receivers mounted on the ceiling in a square grid 1.2m apart. These receivers are connected by a wired network to a central controller. When a bat is

to be located, a radio message is sent containing the tag ID. Then the tag emits an ultrasound signal, that is received by a number of ceiling receivers. For each of these receivers, the controller computes the distance from the bat using time of flight of the signal. Using multilateration, the location of the bat is estimated. Since the radio message travels much faster than the ultrasonic signal, the radio message can be used to synchronize the bat with the infrastructure for the time-of-flight measurements.

The precision of a single measurement is about 14cm in 95% of the time. The average of 10 measurements gives an accuracy of 8cm 95% of the time.

The use of the active bats localization system for sensor networks is limited due to the excessive hardware infrastructure.

5.2.2 Centralized Localization for Sensor Networks

In this section we consider centralized localization algorithms for sensor networks, where the location estimation is performed by a central computer. These approaches generally suffer form limited scalability.

Convex Position Estimation

Convex position estimation [30] considers a problem setup where a number of anchor nodes with known locations and client nodes with unknown positions are given. Any node may define a constraint on the positions of other nodes. In particular, the constraints depicted in Figure 5.1 (b) and (f) are considered. Convex position estimation is a centralized approach for finding location estimates for the nodes with unknown locations that satisfy all given constraints.

The approach taken is to derive a semidefinite program (SDP) that represents all the constraints. An SDP is a relaxation of a linear program, where non-linear constraints can be given in the form $F(x) = F_0 + x_1F_1 + \ldots + x_nF_n \prec 0$, such that $F(x)$ is a negative definite matrix (i.e., all Eigenvalues of $F(x)$ are negative), F_i are symmetric matrices, and x is the solution vector. Such constraints are called called a linear matrix inequality (LMI). Efficient solvers based on interior-point methods exist for such SDP.

Convex position estimation proceeds by mapping each given convex constraint to an LMI. By solving the resulting SDP without specifying an objective function (that would have to be optimized), a solution is found for every unknown node location that satisfies all given constraints. As an alternative, the SDP can be solved four times per client node with different objective functions to find a bounding box that contains the possible solutions for each unknown node location. The center of this bounding box may then be used as a location estimate.

In a random network with average node degree 5.8, the above approach with a distance-bound constraint for each pair of connected nodes provides an average precision in the order of the communication range if 10% of the nodes are anchors with known positions.

Multidimensional Scaling

The algorithm described in [93] considers a setup similar to convex position estimation, where distance estimates are given for pairs of neighboring nodes. Some nodes are anchors with known locations. Location estimates for the remaining nodes are sought that approximately satisfy the given constraints.

The algorithm consists of three steps. Firstly, shortest paths between all pairs of nodes are computed to derive distance constraints for pairs of nodes that are not network neighbors. Secondly, multidimensional scaling is applied to find an assignment of locations to nodes that approximates the given distance constraints. Thirdly, a coordinate transformation is applied to match the given locations of anchor nodes.

Multidimensional scaling is a technique that first finds an assignment of node locations in an m-dimensional space, where $m > 3$. Then, a transformation is applied to map these locations to three-dimensional space, such that the distance relationships are retained.

The approach was examined in a network with a distance constraint for each pair of connected nodes and with 5% of the nodes being anchors. The average precision varied between 1.5 times the communication range for average node degree 6 and 0.5 times the communication range for average node degree 12.

Spring Relaxation

The system reported in [41] consists of an ad hoc infrastructure formed by randomly deployed PDAs and sensor nodes that should be located. Localization is based on acoustic time-of-flight measurements, supported by an explicit time synchronization mechanism. In a first step, PDAs are clustered as depicted in Figure 5.3 (b). PDAs then measure pairwise distances and establish a local coordinate system in each cluster. Nodes that are part of multiple clusters compute a coordinate transformation for adjacent clusters. After the setup has been performed, sensor nodes can emit acoustic signals that are used by a nearby PDA cluster for time-of-flight measurements. From the resulting distance estimates, the location of the node in the cluster's coordinate system is computed and sent to the sensor node.

Setup of the coordinate system and node localization is accomplished by a centralized algorithm that simulates the relaxation of a "spring and mass" model. For this, nodes are mapped to "masses". A distance constraint between two nodes is mapped to a spring between the two masses with a length that is proportional to the distance. The algorithm then finds an assignment of locations to the masses that minimizes the total energy of the system.

In a room-scale experiment (about 10m $\times$ 10m $\times$ 3m), the authors report a precision of about 20cm in 95% of the time.

5.2.3 Distributed Localization for Sensor Networks

In this section we consider distributed localization algorithms for sensor networks, where the location estimation is not performed by a centralized component.

Single-Hop Centroids

This algorithm [17] assumes the existence of anchor nodes that broadcast their locations regularly. A node estimates its location to be the centroid of the locations of the anchor nodes it can hear.

In an experiment, four anchors with a communication range of about 9m were placed at the corners of a 10m $\times$ 10m grid. A node was then placed at the 121 corners of a 1m $\times$ 1m overlay grid to examine the precision of the approach. It was found that the precision is better than 4m in 95% of the time.

Ad Hoc Positioning System (APS)

This algorithm [71] is based on the existence of anchors and supports three modes of operation called *DV-HOP*, *DV-DISTANCE*, and *EUCLIDEAN*. With DV-HOP, anchors flood their locations through the network, such that each node can obtain its hop distance from each anchor. Anchors will obtain hop distances to all other anchors and can thus compute the average hop length by dividing the Euclidean distance between anchors by the number of hops that separate them. The average hop length is also flooded through the network. Each node can now compute an estimate of its distance from each anchor by multiplying the hop distance with the average hop length. Multilateration is then used to compute an estimate of the location of a node. DV-DISTANCE is similar to DV-HOP, but instead of counting hops, the distances between adjacent nodes are measured. The distance of a node from an anchor is then estimated by the sum of the distances along the shortest path. Again, multilateration is used to obtain a location estimate. With the EUCLIDEAN approach, nodes that have a sufficient number of neighbors with known location and distance perform multilateration to estimate its location. This process is iterated until all nodes have estimated their locations.

The authors examined the performance of the proposed approaches by simulating of a network with 100 randomly placed nodes with an average node degree of 7.6. The average precision of DV-HOP varied between 0.5 times the communication range for 10% anchors and 0.2 times the communication range for 90% anchors. The precision of the DV-DISTANCE and EUCLIDEAN methods heavily depends on the precision of the distance measurements. The precision of DV-DISTANCE varied between 1.2 times the communication range for 90% average distance precision and 10% anchors and 0.2 times the communication range for 2% average distance measurements precision and 10% anchors. The precision of EUCLIDEAN varied between the communication range for 90% average distance precision and 10% anchors and 0.2 times the communication range for 2% average distance measurement precision and 10% anchors.

Iterative Multilateration

This algorithm [89] is conceptually similar to the EUCLIDEAN variant of APS. Initially, only the locations of anchor nodes are known. Client nodes with unknown positions use multilateration to at least three neighbors with known locations. After a node has estimated its locations, it can also be used as a reference in subsequent iterations of the algorithm. In some situations, a node may not have a sufficient number of neighbors with known locations. In this case, collaborative multilateration (cf. 5.1.3) may be used.

This algorithm is very sensitive to the number and placement of the anchors. The authors performed simulations to characterize the effect of the anchor infrastructure on the percentage of nodes that can successfully estimate their location. In a rectangular area of size 100 by 100 units, 200 (300) sensor nodes with communication range 10 were randomly distributed. In order to allow at least 90 % of the nodes to estimate their locations, 45 % (10 %) of the nodes must be anchors.

Collaborative Multilateration

This algorithm [90] is an extension of earlier work [89] and consists of four phases. In the first phase, the network is clustered such that each cluster contains anchors and client nodes, where anchors and distances between nodes define a unique location for each client node (cf. Figure 5.2). In other words, collaborative multilateration as discussed in Section 5.1.3 can be used to find unique locations for all client nodes in such a cluster. Anchors may participate in multiple clusters. "Underconstrained" client nodes do not participate in any cluster, but will be handled in the fourth phase.

In the second phase, an initial location estimate is computed for each client node. For this, distances between nodes are measured and shortest paths between each pair of (node, anchor) are computed. For each of these pairs, the length of the shortest path (i.e., sum of the measured distances along the path) is used to construct a bounding box on the location of the respective node. The node is located at the center of the intersection of all its bounding boxes.

In the third phase, the initial location estimates are further refined. There is a centralized and a distributed version of this phase. With the centralized version, a cluster head is elected per cluster which solves the collaborative multilateration problem for its cluster (cf. Section 5.1.3). The solution is derived by iteratively applying a Kalman filter to the initial location estimates, such that the value of the objective function of the optimization problem is reduced in each step. With the distributed version, each node uses the initial location estimates of its neighbors to perform multilateration, resulting in a refined location estimate. This process is repeated until the distance between old and updated location falls below a given threshold. These updates are performed in a fixed sequence on the nodes to prevent them from getting stuck in local minima.

In the fourth phase, locations of underconstrained nodes are further refined. For this, the client nodes with known location estimates (that were computed in

phase three) are treated as additional anchors. Using the distributed approach of the third phase, the locations of underconstrained nodes are then refined.

The algorithm was evaluated by simulation. The communication range was set to 15m and distance measurement errors were modeled by a zero-mean Gaussian random variable with a standard deviation of 2cm. With 6 anchors, an average node degree of 6, the number of client nodes was varied between 10 and 100. The average precision was 2.8cm with a standard deviation of 1.6cm.

Hop-Terrain

This algorithm [88] also assumes that a certain percentage of nodes are anchors. The algorithm consists of a start-up phase and a refinement phase. In the start-up phase, all anchors flood the network to enable all nodes to estimate their hop distance from all anchors. Anchors can then compute the average Euclidean length d of one hop by dividing the known Euclidean distance between two anchors by the number of hops separating them. This average hop length is also flooded through the network. Eventually, every node can come up with an estimate of its distance from all anchors by multiplying the hop count with the average hop length. Using multilateration as described in Section 5.1.3, each node ends up with a first estimate of its position.

In the refinement phase, each node repeatedly measures distances to its neighbors and performs multilateration to obtain a refined location estimate. This process is repeated until the distance between old and updated location falls below a given threshold. As an extension, each node can assign a confidence to its location estimate which is calculated from the number of neighbors (more neighbors likely result in more precise location estimates) and from the local network topology. The confidence is used to weight the significance of each constraint in the error metric of the optimization problem.

In simulations with a random network topology with 5% anchors and 5% average range measurement error, the precision varied between 0.3 times the communication range for an average node degree of 7 and 0.1 times the communication range for an average node degree of 15. For average node degrees below 7, large fractions of nodes remain without a location estimate, since they have less then 4 neighbors.

Precision-Based Iterative Multilateration

The structure of this algorithm [36] is similar to the work presented in [89]. However, besides a location estimate, this algorithm also calculates and uses the standard deviation of the estimated location as a measure of the achieved precision. Also, if a node does not have a sufficient number of neighbors with known positions, the distance to a remote reference node (and standard deviation) is estimated by the length of shortest path across multiple hops. After nodes have obtained an initial location estimate, these estimates are further improved in a refinement phase similar to the ones described in [88, 90].

The authors performed simulations to examine the performance of the proposed algorithm. In a square area of 15 units edge length, 225 nodes with radio range 2.1 units were randomly placed with 5 % of them being anchors. The standard deviation of the range measurement error was 20 % of the distance. With this setup, the average location error was about 17 % with a standard deviation of 23 %.

Self-Positioning Algorithm (SPA)

SPA [20] does not require anchors. In a first phase, each node measures distances to its neighbors and broadcasts these distances to its neighbors. After this, each node knows the distance to each of its neighbors and the distances between some pairs of its neighbors. Then each node constructs a local coordinate system using two of its neighbor nodes as discussed in Section 3.3.1. In the second phase, coordinate transformations are computed between the coordinate systems of adjacent nodes. In the third phase, a global coordinate system is selected and coordinate transformations are computed to transform the local coordinate systems to this global system. For this, a set of nodes called the Location Reference Group (LRG) is elected such that the degree of mobility of the centroid of these nodes is small in order to avoid frequent adjustments of the global coordinate system. The global coordinate system is then defined by the average of the local coordinate systems of the nodes in the LRG (i.e., origin is centroid of the origins of the individual coordinate systems, axis vectors are averages of the axis vectors of the individual coordinate systems).

We are not aware of any results about the precision this approach can provide.

APIT

This algorithm [44] assumes the existence of anchor nodes and is based on the triangle test described in Section 5.1.3. For various triangles formed by combinations of three anchors, the triangle test is performed to decide whether a node is inside or outside this triangle. The largest intersection of "inside" triangles for the node is then computed using the pixel approximation described in Section 5.1.3. The center of gravity of the intersection region is used as the location estimate for the considered node.

To implement this approach, anchors broadcast their ID and location. Each node records the ID, location, and received signal strength for all anchors it can hear. The resulting anchor table is then broadcast, so that each node knows its own table and the tables of all neighbors. Each node then locally performs the triangle test for all combinations of three anchors based on these tables and computes an estimate of its own location.

In simulations it was shown that the precision of this approach mainly depends on the number of anchors a node can hear, which in turn depends on the number of anchors, their distribution, and the communication range of the anchors. In a random network with an average node degree of 8 and where the communication

range of anchors is 10 times the communication range of nodes, the average precision varies between 7 times the node communication range (if a node can hear 3 anchors on average) and 0.5 times the communication range (if a node can hear 25 anchors on average).

5.3 Problem Statement

Our goal is the provision of a scalable and resource-efficient localization approach for Smart Dust (cf. Section 2.5.3). As we show below, this setup represents a point in the design space that is not covered by existing approaches.

5.3.1 Device Challenges

The physical characteristics of Smart Dust particles represent a major challenge to localization approaches. In particular, the available resources for communication, computation, storage, and energy are severely limited by the small size of these devices. Hence, the overhead of a localization algorithm must be rather low with respect to these resources.

These resource limitations are also the reason for the passive optical communication approach adopted by Smart Dust. While traditional radio-based sensor nodes can actively communicate with its network neighborhood, Smart Dust particles can only communicate with a base station. Since communication is directed, interaction with the base station is only possible if the base station happens to point its laser beam towards a particular Smart Dust particle.

Most of the localization systems discussed earlier assume direct node-to-node communication, which is not possible with Smart Dust. In order to match the stringent resource constraints of Smart Dust, a prospective localization system would ideally reuse the existing hardware components (i.e., optical transceiver) instead of requiring additional facilities.

5.3.2 Resource Efficiency

As discussed in Section 3.2.5, a key to achieving resource efficiency is the exploitation of scope and lifetime requirements of the application. In other words, localization should be performed only where and when required by the application. Hence, it would be desirable that each Smart Dust particle could estimate its location on demand, independent of other particles.

Note that with almost all distributed localization approaches for sensor networks discussed in Section 5.2, estimating the location of a single node requires support from a large number of other nodes in the network. Hence, these approaches are not well-suited for providing localization on demand.

5.3.3 Minimal Infrastructure

Since Smart Dust is envisioned to be deployed in hostile or unexploited areas, it is rarely possible to install extensive hardware infrastructure besides the single base station, which is required for communication. Hence, a localization scheme for Smart Dust can make use of and extend the existing base station, but should not require the installation of additional, geographically dispersed infrastructure elements.

5.3.4 Scalability

It is anticipated that large numbers of Smart Dust particles will be densely deployed for detailed observation of a target area. However, it is likely that portions of such a deployment are less dense, and some deployments may consist of small numbers of nodes. A localization approach should ideally support this wide spectrum of deployments ranging from a single node or few nodes to dense networks of thousands of nodes.

This requirement represents a major challenge as indicated by the algorithms discussed in Section 5.2, where many approaches require a certain minimum network density to provide a reasonable precision.

Also, large networks require that there is no per-node overhead such as for the calibration of individual nodes. Moreover, centralized approaches should be avoided since they represent a scalability bottleneck for large networks. Rather, Smart Dust particles should be able to compute their locations on their own instead of relying on external infrastructure for computing location estimates.

5.4 The Lighthouse Location System

This section presents the Lighthouse Location System for Smart Dust. In order to point out the basic ideas behind this system, we will first examine a simplified idealistic system. This examination will be followed by a more thorough discussion of a realistic system that can actually be built. We will go on by presenting a prototype implementation, a set of measurements, and an evaluation of several aspects of the system.

5.4.1 An Idealistic System

Consider the special lighthouse depicted in Figure 5.4, which has the property that the emitted beam of light is parallel with a constant width b when seen from top. When seen from the side, the angle of beam spread of the parallel beam is large enough so that it can be seen from most points in space.

When this parallel beam passes by an observer, he will see the lighthouse flash for a certain period of time t_{beam}. Note that t_{beam} depends on the observer's distance d from the rotation axis of the lighthouse since the beam is parallel. Assuming the

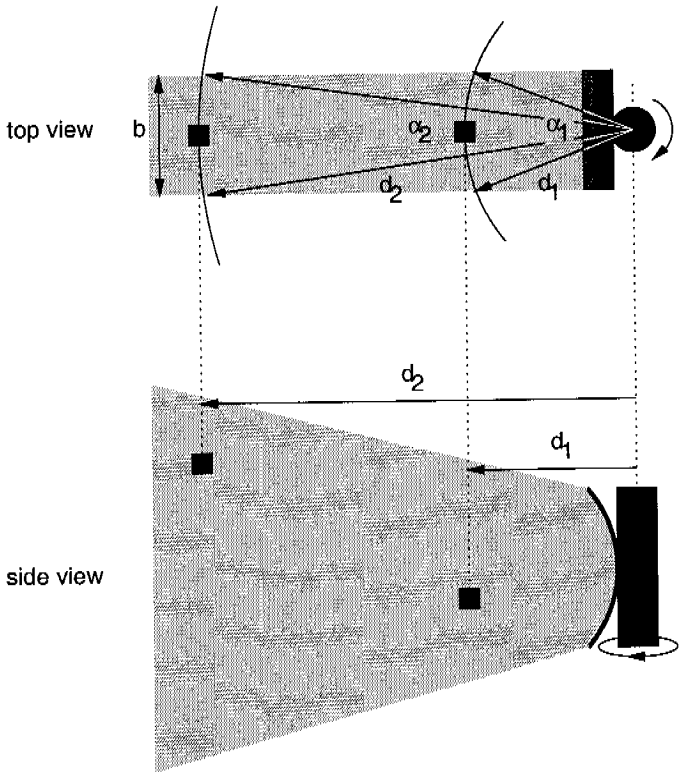


Figure 5.4: Top and side view of an idealistic lighthouse with a parallel beam of light.

lighthouse takes t_{turn} for a complete rotation, we can express the angle α, under which the observer sees the beam of light as follows:

$$\alpha = 2\pi \frac{t_{\text{beam}}}{t_{\text{turn}}} \tag{5.10}$$

Figure 5.4 shows two observers (depicted as squares) at distances d_1 and d_2 and the respective angles α_1 and α_2. We can express d in terms of α and the width b of the beam as follows:

$$d = \frac{b}{2\sin(\alpha/2)} \tag{5.11}$$

By combining Equations 5.10 and 5.11 we obtain the following formula for d in terms of b, t_{beam}, and t_{turn}:

$$d = \frac{b}{2\sin(\pi t_{\text{beam}}/t_{\text{turn}})} \tag{5.12}$$

Note that the distance d obtained this way is the distance of the observer to the lighthouse rotation axis as depicted in the side view in Figure 5.4. That is, all the points in space with distance d form a cylinder (not a sphere!) with radius d centered at the lighthouse rotation axis.

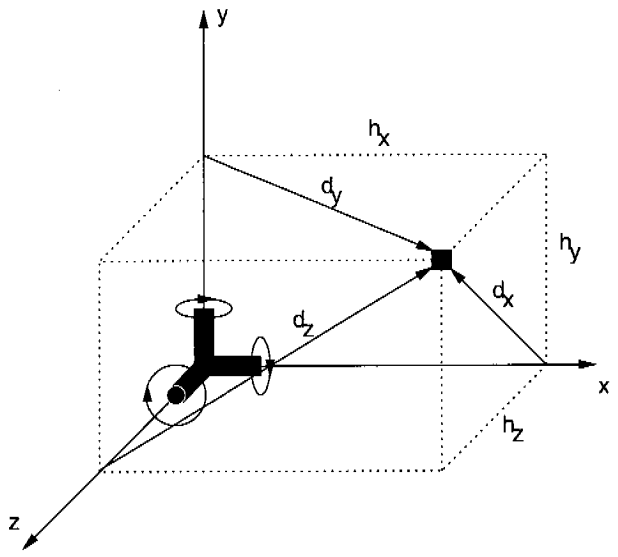


Figure 5.5: 3D Localization support device consisting of three mutually perpendicular lighthouses.

Based on the above observations, we can build a simple ranging system consisting of a lighthouse and an observer. The observer device contains a photo detector and a clock. When the photo detector first sees the light it records the corresponding point in time t_1. When the photo detector no longer sees the light it records t_2. When it sees the light again it records t_3. With $t_{\text{beam}} := t_2 - t_1$ and $t_{\text{turn}} := t_3 - t_1$ the observer can apply Equation 5.12 in order to calculate its distance d from the lighthouse rotation axis. Note that if t_{turn} is constant it has to be measured only once since it does not change with distance. Also note that the necessary hardware resources of the observer device are matched by a Smart Dust node as explained in Section 2.5.3.

This ranging scheme can be used to build a *single* beacon device, which allows observers to autonomously determine their position relative to the beacon in space. This beacon device consist of three lighthouses with mutually perpendicular rotation axes as depicted in Figure 5.5. Assuming an observer measures the distances d_x, d_y, and d_z as indicated above, its location can be determined by computing the intersection point(s) of three cylinders with radius d_x, d_y, d_z centered at the respective lighthouse rotation axes. Note that there are 8 such intersection points in general, one in each of the 8 quadrants of the coordinate system. If we can ensure, however, that all observers are located in a single quadrant (e.g., the main quadrant defined by the points (h_x, h_y, h_z) with $h_x, h_y, h_z \geq 0$), there is a unique intersection point. This intersection point can be obtained by solving the following equation system for h_x, h_y, h_z:

$$\begin{aligned} d_x^2 &= h_y^2 + h_z^2 \\ d_y^2 &= h_x^2 + h_z^2 \\ d_z^2 &= h_x^2 + h_y^2 \end{aligned} \tag{5.13}$$

Note that this equation system does not necessarily have a solution, since the values d_x, d_y, d_z are only approximations obtained by measurements. If there is no solution, an approximation for the intersection point can be obtained using minimum mean square error (MMSE) methods. The solution (h_x, h_y, h_z) obtained this way minimizes the sum of the squares of the differences of the left hand and right hand sides of the Equations 5.13. However, if the equation system has a solution, it can be directly solved using the following set of equations, again assuming that the observer is located in the main quadrant of the coordinate system depicted in Figure 5.5:

$$\begin{aligned} h_x &= \sqrt{(-d_x^2 + d_y^2 + d_z^2)/2} \\ h_y &= \sqrt{(d_x^2 - d_y^2 + d_z^2)/2} \\ h_z &= \sqrt{(d_x^2 + d_y^2 - d_z^2)/2} \end{aligned} \tag{5.14}$$

The setup of the complete location system can now be described. The base station is equipped with three mutually perpendicular lighthouses as depicted in Figure 5.5. At startup, the base station broadcasts certain calibration parameters (e.g., the beam width b for each of the lighthouses) to all dust nodes. The latter use a clock to measure the amount of time during which each of the lighthouses beams are visible. Using Equations 5.12 and 5.13, nodes can autonomously compute their location in the reference grid defined by the base station's three lighthouses.

The description of the system's principles gives rise to a number of practical questions. First of all, it is not clear at all whether a system fulfilling the above requirements (e.g., parallel beam) can actually be built in practice. Moreover, we did not discuss the problem how a dust node can distinguish the different beams of the lighthouses, or what happens if a dust node "sees" the beams of two lighthouses at the same time. We will discuss these issues in the next sections in order to lay the foundation for an implementation of the system.

5.4.2 A Realistic System

During first experiments it turned out that actually building a lighthouse with a sufficiently exact parallel beam is very difficult, at least given the limited technical capabilities that were available to us. This has the unfortunate consequence, that the model described in Section 5.4.1 cannot directly be used due to the resulting high inaccuracies. To understand the reason of these inaccuracies, consider the following example, where we assume a beam width of 10cm. Even if the angle of beam spread is only 1° (instead of 0° for an ideal parallel beam), the width of the beam at a distance of 5m would be about 18.7cm, resulting in an error of almost 90%. The relative error could be reduced somewhat by increasing the width of the beam. However, a large beam width also results in a large and clumsy base station device.

Therefore, instead of building a system perfectly matching the requirements of Section 5.4.1, we have to adapt our model to a system which can actually be built.

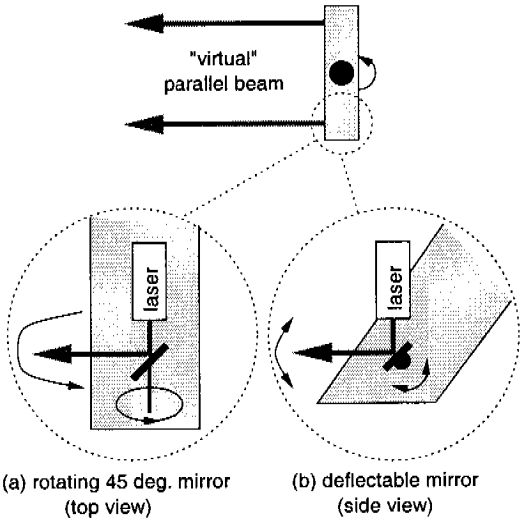


Figure 5.6: A rotating lighthouse with a "virtual" parallel beam whose outline is defined by two parallel laser beams. Rotating (a) or deflectable mirrors (b) are used make the laser beams scan the northern hemisphere of the lighthouse.

In order to develop such a model, we first have to examine ways of generating near-parallel beams.

Beam Generation

In order to keep the hardware and energy overhead on the Smart Dust nodes small, the beam must be easily detectable. Furthermore, the system should work with high accuracy even if the base station is far away (tens of meters, say) from the nodes. Therefore we decided to use a laser-based approach. As mentioned above, the beam should be as wide as possible in order to keep inaccuracies small. In order to achieve this, we use *two* lasers to create the *outline* of a parallel beam as depicted in the upper half of Figure 5.6. This makes no difference to a single wide beam, since we are only interested in the edges of the beam (i.e., change from "dark" to "light" and vice versa) in order to measure t_{beam} and t_{turn}.

Due to the narrow laser beams, the "virtual" parallel beam generated this way can only be seen from a single plane, however. In order to ensure that the beam can be seen from any point in the northern hemisphere of the lighthouse without defocusing the lasers, the laser beams have to scan this space in some way. The lower half of Figure 5.6 depicts two ways to achieve this. The first approach uses a small mirror mounted on a rotating axle under an angle of 45°. By pointing the laser at this mirror, the reflected rotating beam describes a plane. With commercial off the shelf technology we can easily achieve a rotation frequency of about 300Hz. The second approach uses a small deflectable MEMS mirror similar to the one used as part of the corner cube retroreflector (CCR). Figure 5.7 shows such a device [25], which operates at 35kHz and achieves a deflection angle of 25°. A laser beam pointed at such a mirror can thus sweep over an angle of 50° at a frequency

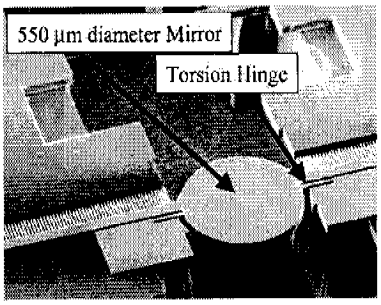


Figure 5.7: MEMS deflectable mirror with maximum deflection angle of 25° and resonant frequency of 35kHz [7].

of 35kHz.

Based on this approach, a lighthouse consists of a (slowly) rotating platform, on which two semiconductor laser modules and two rotating (or deflectable) mirrors are mounted. However, as mentioned at the beginning of Section 5.4.2, it is next to impossible to assemble all the pieces such that the resulting "virtual" wide beam is almost parallel. Therefore, we have to come up with a model which describes an imperfect but realistic system. The model discussed below is based on rotating mirrors, since we used this approach in our prototype implementation of the system. However, the model equally applies to a system based on deflectable mirrors.

The Lighthouse Model

We use Figure 5.8 to explain the lighthouse model. It shows a simplified top and side view of the lighthouse. Each view shows the two mirror's rotation axes and the corresponding reflected rotating laser beams. Note that in general the angle enclosed by the mirror rotation axis and the mirror will not be exactly 45° (i.e., $\beta_i \neq 0°$) due to manufacturing limitations. Therefore, the rotating reflected laser beams will form two cones as depicted in Figure 5.8. Moreover, the two mirror's rotation axes will not be perfectly aligned. Instead, the dashed vertical line (connecting the apexes of the two cones formed by the rotating laser beams) and the mirror rotation axes will enclose angles γ_i in the side view and angles δ_i in the top view that are different from 0°. Additionally, the figure shows the rotation axis of the lighthouse platform and its distances b_1 and b_2 to the apexes of the two light cones. The *lighthouse center* is defined as the intersection point of the lighthouse platform rotation axis and the dashed vertical line in Figure 5.8. Note that the idealistic lighthouse described in Section 5.4.1 is a special case of this more complex model with $\beta_i = \gamma_i = \delta_i = 0°$ and $b_1 = b_2$.

Now let us consider an observer (black square) located at distance d from the main lighthouse platform rotation axis and at height h over the lighthouse center. We are interested in the width b of the virtual wide beam as seen by the observer. Let us assume for this that we can build a lighthouse with $b_1 \approx b_2$ and $\beta_i, \gamma_i, \delta_i \approx 0°$, i.e., we do our best to approximate the perfect lighthouse described in Section 5.4.1. Then we can express b approximately as follows:

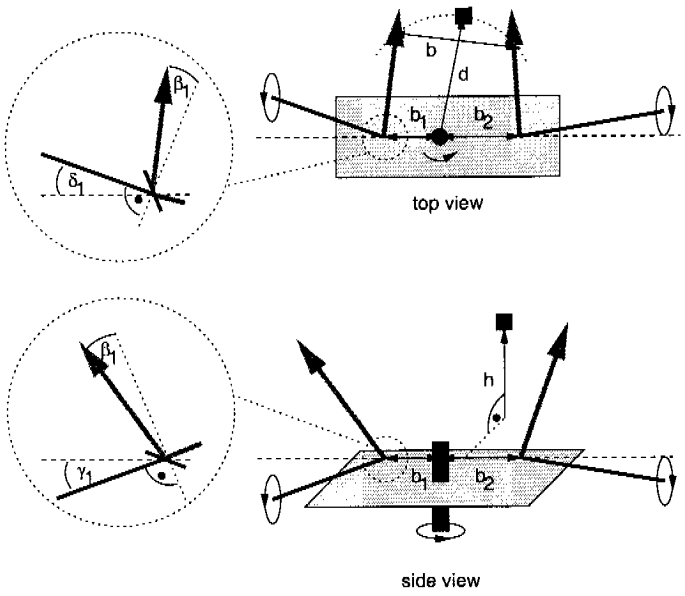


Figure 5.8: Model of a realistic lighthouse based on rotating mirrors. The zoom-ins show detail for one rotating mirror in the top and side views. The other rotating mirror has respective parameters β_2, γ_2, and δ_2.

$$\begin{aligned} b \;\approx\; & b_1 + b_2 + \sqrt{d^2 + h^2}(\sin\beta_1 + \sin\beta_2) + \\ & h(\tan\gamma_1 + \tan\gamma_2) + d(\sin\delta_1 + \sin\delta_2) \end{aligned} \tag{5.15}$$

The inaccuracy results from the last two terms, which are linear approximations of rather complex non-linear expressions. For $\beta_1 = \beta_2 = 0°$, however, expression 5.15 becomes an equation. We will allow these factors to be built into the error term.

With $C^b := b_1 + b_2$, $C^\beta := \sin\beta_1 + \sin\beta_2$, $C^\gamma := \tan\gamma_1 + \tan\gamma_2$, and $C^\delta := \sin\delta_1 + \sin\delta_2$ we can rewrite expression 5.15 as

$$b \approx C^b + \sqrt{d^2 + h^2}C^\beta + hC^\gamma + dC^\delta \tag{5.16}$$

Note that C^b, C^β, C^γ, and C^δ are fixed lighthouse parameters. We will show below how they can be determined using a simple calibration procedure. We can express b also in terms of the angle α obtained using Equation 5.11:

$$b = 2d\sin\frac{\alpha}{2} \tag{5.17}$$

Combining expressions 5.16 and 5.17 we obtain the following expression which defines the possible (d, h) locations of the observer, given a measured angle α and the lighthouse calibration values C^*:

$$2d\sin\frac{\alpha}{2} \approx C^b + \sqrt{d^2 + h^2}C^\beta + hC^\gamma + dC^\delta \tag{5.18}$$

Note that for given C^* and α the points in space whose d and h values are solutions of Equation 5.18 form a rotational hyperboloid centered at the rotation axis of the

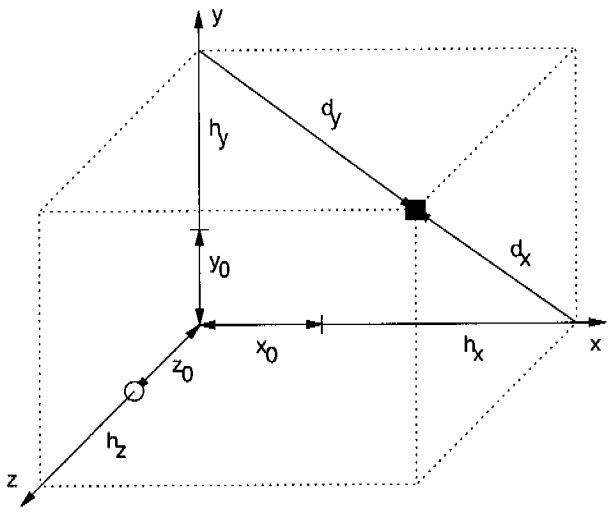


Figure 5.9: Positions of the lighthouses in the coordinate system. d_z is not shown for clarity.

lighthouse. In the special case $\beta_i = \gamma_i = \delta_i = 0°$ and $b_1 = b_2$ this hyperboloid becomes a cylinder as in the idealistic model described in Section 5.4.1.

Location Computation

Similar to the idealistic model described in Section 5.4.1, the location of the observer can be obtained by determining the intersection point(s) of the three rotational hyperboloids defined by Equation 5.18. However, since the observed virtual beam width b now additionally depends on the height h of the observer, we have to take into account the exact positions of the three lighthouses. Figure 5.9, which shows an extended version of Figure 5.5, illustrates this. The marks on the coordinate axes show the positions of the lighthouse center (as defined in Section 5.4.2) of each of the three lighthouses. That is, the coordinates of the observer are $(x_0+h_x, y_0+h_y, z_0+h_z)$ with respect to the origin formed by the intersection of the three lighthouse rotation axes. In order to obtain approximations for the values h_x, h_y, and h_z, we have to solve the following equation system:

$$\begin{aligned}
2d_x \sin\tfrac{\alpha_x}{2} &= C_x^b + \sqrt{d_x^2 + h_x^2}C_x^\beta + h_x C_x^\gamma + d_x C_x^\delta \\
2d_y \sin\tfrac{\alpha_y}{2} &= C_y^b + \sqrt{d_y^2 + h_y^2}C_y^\beta + h_y C_y^\gamma + d_y C_y^\delta \\
2d_z \sin\tfrac{\alpha_z}{2} &= C_z^b + \sqrt{d_z^2 + h_z^2}C_z^\beta + h_z C_z^\gamma + d_z C_z^\delta \\
d_x^2 &= (y_0 + h_y)^2 + (z_0 + h_z)^2 \\
d_y^2 &= (x_0 + h_x)^2 + (z_0 + h_z)^2 \\
d_z^2 &= (x_0 + h_x)^2 + (y_0 + h_y)^2
\end{aligned} \tag{5.19}$$

The indices $\{x, y, z\}$ indicate which lighthouse the values are associated with. As with Equation system 5.13, this system does not necessarily have a solution, since the parameters are only approximations obtained by measurements. Therefore, minimum mean square error (MMSE) methods have to be used to obtain approx-

imations for the h_*. However, if the equation system 5.19 has a solution, we can approximately solve it by simple iteration. For this, we first transform each of the six equations of equation system 5.19 in order to obtain the following fixpoint form:

$$\begin{aligned} h_x &= f_1(d_x) \\ h_y &= f_2(d_y) \\ h_z &= f_3(d_z) \\ d_x &= f_4(h_y, h_z) \\ d_y &= f_5(h_x, h_z) \\ d_z &= f_6(h_x, h_y) \end{aligned} \tag{5.20}$$

Note that we did not show arguments of the f_i (i.e., C_*^*, α_*, x_0, y_0, z_0) that do not change during iterative evaluation of the equation system. By using appropriate values for h_x^0, h_y^0, h_z^0, and Δ, we can obtain approximate solutions for h_x, h_y, h_z with the following algorithm:

$h_x := h_x^0$;
$h_y := h_y^0$;
$h_z := h_z^0$;
while (true) {
    $h'_x := f_1(f_4(h_y, h_z))$;
    $h'_y := f_2(f_5(h_x, h_z))$;
    $h'_z := f_3(f_6(h_x, h_y))$;
    if $(|h'_x - h_x| + |h'_y - h_y| + |h'_z - h_z| < \Delta)$
        break;
    $h_x := h'_x$;
    $h_y := h'_y$;
    $h_z := h'_z$;
}

At first, the h_* are initialized to the start values h_*^0. Using the f_i, new approximations h'_* are computed. We are finished if the new values are reasonably close to the original h_*. Otherwise we update the h_* to the new values and do another iteration. For good convergence of this algorithm the partial derivatives of the $f_i \circ f_{3+i}$ in the environment of the solution (h_x, h_y, h_z) should be small, which is typically true. In our prototype implementation we use $h_*^0 := 100\text{cm}$ and $\Delta := 0.1\text{cm}$. With this configuration, the algorithm typically performs 4-6 iterations.

Calibration

What remains to be shown is how we can obtain values for x_0, y_0, z_0, and C_x^*, C_y^*, C_z^*. Since the values x_0, y_0, z_0 are uncritical for the achieved accuracy, we assume they are measured directly. The C_*^* values, however, are very critical for the accuracy as was shown with the example at the beginning of Section 5.4.2. Therefore we have to perform a calibration.

For each of the three lighthouses we have to determine values for the four variables $C^b, C^\beta, C^\gamma, C^\delta$. For this, we place the observer at known locations (d_i, h_i) and obtain the respective α_i using Equation 5.10. Doing so for at least four locations and using equation 5.18, we obtain the following linear equation system in $C^b, C^\beta, C^\gamma, C^\delta$:

$$\begin{aligned} 2d_1 \sin\tfrac{\alpha_1}{2} &= C^b + \sqrt{d_1^2 + h_1^2}C^\beta + h_1 C^\gamma + d_1 C^\delta \\ 2d_2 \sin\tfrac{\alpha_2}{2} &= C^b + \sqrt{d_2^2 + h_2^2}C^\beta + h_2 C^\gamma + d_2 C^\delta \\ 2d_3 \sin\tfrac{\alpha_3}{2} &= C^b + \sqrt{d_3^2 + h_3^2}C^\beta + h_3 C^\gamma + d_3 C^\delta \\ 2d_4 \sin\tfrac{\alpha_4}{2} &= C^b + \sqrt{d_4^2 + h_4^2}C^\beta + h_4 C^\gamma + d_4 C^\delta \end{aligned} \tag{5.21}$$

As with the other equation systems, this system does not necessarily have a solution, since the parameters are only approximations obtained by measurements. If the system has a solution, it can be obtained by Gaussian elimination. For this, the d_i and h_i have to fulfill certain requirements. One simple rule of thumb is that both the d_i and the h_i should be pairwise distinct.

Better results can be obtained if the system is calibrated at more than four points. The resulting over-determined equation system can than be solved using MMSE methods in order to obtain approximations for the C^*. Formally, we can rewrite Equation system 5.21 as

$$\begin{pmatrix} 1 & U_1 & V_1 & W_1 \\ 1 & U_2 & V_2 & W_2 \\ 1 & U_3 & V_3 & W_3 \\ \vdots & \vdots & \vdots & \\ 1 & U_n & V_n & W_n \end{pmatrix} \begin{pmatrix} C^b \\ C^\beta \\ C^\gamma \\ C^\delta \end{pmatrix} = \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ \vdots \\ T_n \end{pmatrix} \tag{5.22}$$

with $T_i = 2d_i sin(\frac{\alpha_i}{2})$, $U_i = \sqrt{d_i^2 + h_i^2}$, $V_i = h_i$, $W_i = d_i$. Rewriting 5.22 as $Ax = b$, we can solve the least squares problem by solving the following equation for x:

$$A^T A x = A^T b \tag{5.23}$$

which is equivalent to solving the following equation for $(C^b, C^\beta, C^\gamma, C^\delta)$ using Gaussian elimination:

$$\begin{pmatrix} \sum 1 & \sum U_i & \sum V_i & \sum W_i \\ \sum U_i & \sum U_i^2 & \sum U_i V_i & \sum U_i W_i \\ \sum V_i & \sum V_i U_i & \sum V_i^2 & \sum V_i W_i \\ \sum W_i & \sum W_i U_i & \sum W_i V_i & \sum W_i^2 \end{pmatrix} \begin{pmatrix} C^b \\ C^\beta \\ C^\gamma \\ C^\delta \end{pmatrix} = \begin{pmatrix} \sum T_i \\ \sum U_i T_i \\ \sum V_i T_i \\ \sum W_i T_i \end{pmatrix} \tag{5.24}$$

All summations run over $i = 1...n$.

Note that calibration has to be performed only once for each base station (assuming that the system is stable enough and needs not be re-calibrated) and is independent of the receiver nodes. Therefore, calibration can be performed using a

more powerful receiver device than the limited Smart Dust node. As explained in Section 5.4.1, the base station broadcasts these calibration parameters to the Smart Dust nodes, which use them to compute their location using Equation System 5.19.

5.4.3 Prototype Implementation

In order to evaluate the concepts developed in Section 5.4.2, we implemented a prototype system. To keep the hardware overhead small, this prototype system consists of only two lighthouses and allows observers located on the plane $y = 0$ to determine their x and z coordinates. From a conceptual point of view, the differences to a 3D system are minimal.

The Base Station

Figure 5.10 shows a picture of the prototype base station. It consists of two mutually perpendicular lighthouses. The main lighthouse platform takes about $t_{\text{turn}} = 60\text{s}$ for one rotation. The platform is driven by a geared electro motor manufactured by FTB [120], which has a low flutter of about 0.1% of the rotation speed. Using an LM317 [124] adjustable voltage regulator, the voltage supply of the motor and thus the rotation speed of the platform can be adjusted. The two bars that extend from under the platform are used to move the center of gravity of the platform to the rotation axis, such that the platform rotates at a constant speed.

The power supply for the rotating platform is implemented by a stereo jack and associated plug. While the plug is fixed to the axle of the rotating platform, the jack is affixed to the chassis using a thin steel wire. This way, the round plug can rotate in the jack.

Beam generation is based on rotating mirrors as described in Section 5.4.2. Both rotating mirrors are driven by a single Graupner SPEED 280 electro-motor. In order to reduce vibrations, we did not use a rigid axle to connect the mirrors to the motor. Instead, we used small steel springs as axles. The rotating mirrors are supported by two ball bearings each. Two 1mW 650nm semiconductor laser modules with adjustable focus point their beam at the rotating mirrors.

The supply voltage of the motor and thus its rotation speed can be adjusted using an LM317 voltage regulator. The mirror rotation speeds of the two lighthouses are slightly different ($t_{\text{mirror}} = 4\text{ms}$ and $t_{\text{mirror}} = 5\text{ms}$ for one rotation, respectively), such that the observer can distinguish the two lighthouses based on the time interval between successive light flashes, which will be explained in more detail in Section 5.4.3. Hence, in order to detect a beam, the observer's photo detector must at least be hit twice by the rotating laser beam. Note that due to the fast rotation of the laser beams, the average light intensity is low enough to be eye-safe.

There is a slight chance that the photo detector is hit by the beams of both lighthouses at the same time. We will explain in Section 5.4.3 how an observer can detect and handle this situation. However, since the diameter of the laser beams is rather small, the likelihood of this event is small. By selecting slightly

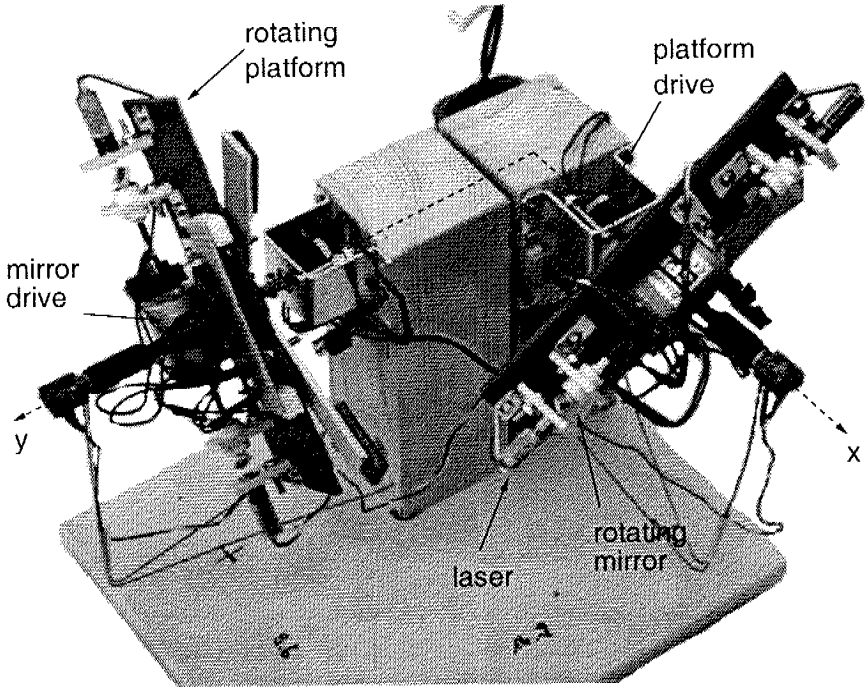


Figure 5.10: Prototype base station consisting of two lighthouses and the resulting 2D coordinate system.

different platform rotation speeds for the two lighthouses, we can ensure that for each observer this happens only once in a while. In our experiments this happened about every 100 lighthouse rotations at a single fixed observer.

The whole device is powered by a 7.2V 2400mAh nickel-cadmium battery, which lasts for about 3 hours, which equals about 3 watts per lighthouse. Note however, that almost all of the power is consumed by the motors driving the mirrors, which operate at 6 volts. Moreover, the voltage converters burn quite some amount of power, since the lasers and the platform drives operate at 3 volts. We expect a low voltage design using deflectable MEMS mirrors instead of rotating mirrors to consume as few as 0.3 watts per lighthouse, since the MEMS mirrors consume considerably less power than the motors driving the rotating mirrors.

The Nodes

The receiver prototype consists of a small electronic circuit connected to the parallel port of a laptop computer running Linux. Figure 5.11 shows a schematic diagram of the receiver hardware. A photo diode converts the intensity of the incident light into a proportional voltage. The light that is incident to the photo diode mainly consists of three components:

- direct current (DC) components resulting from slowly changing daylight
- low frequency components resulting from artificial lighting powered with 50Hz alternating current (AC)

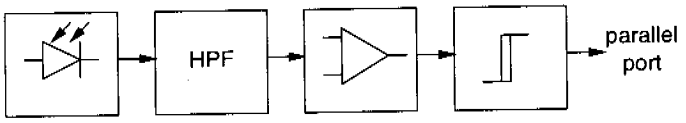


Figure 5.11: Schematic diagram of the receiver hardware.

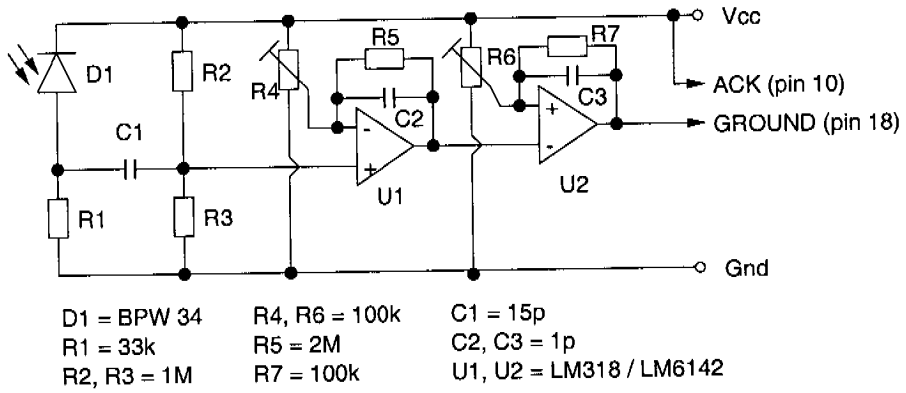


Figure 5.12: Circuit diagram of the receiver hardware.

- higher frequency components resulting from laser light flashes at about 200Hz-300Hz ($1/t_{\text{mirror}}$)

Since we are only interested in the higher frequency laser flashes, we run the output signal of the photo diode through a high pass filter (HPF) which removes DC and low frequency components. Due to this, the detector is insensitive to daylight and artificial light.

The output of the HPF is then amplified using an operational amplifier, whose output is in turn fed into a Schmitt Trigger. The latter implements a hysteresis, i.e., when the input voltage level exceeds a certain value V_1 it lowers the output voltage to a minimum. When the input voltage falls below a certain value V_2, the Schmitt Trigger raises the output voltage to a maximum. The output of the Schmitt Trigger is connected to the parallel port, so that each laser flash on the photo diode causes a parallel port interrupt to be triggered.

Figure 5.12 shows the complete circuit diagram of the device. D1 and R1 form the photo diode stage, C1 and R3 implement the HPF. R2 is used to shift the output from the HPF to about half of the supply voltage as input for the amplifier, which consists of R4 for reference voltage generation, U1, R5, and C1. U2, R7, and C3 implement the Schmitt Trigger, R6 is used to generate the voltage reference for the latter. Since the output of the Schmitt Trigger is inverted, we connect the output to the GROUND pin (pin #18) of the parallel port and Vcc to the ACK pin (pin #10). The LM318 [124] is a high speed operational amplifier commonly available at electronics supply stores. The device is powered by a small 9V battery (connected to Vcc and Gnd), since the used operational amplifiers require a symmetric voltage supply of at least +4.5V and -4.5V. A later version of this design uses the LM6142 [124] low-voltage low-power dual channel operational amplifier and operates at +1.5 and -1.5V. We connected this device to the BTnodes [116] (cf. Chapter 6).

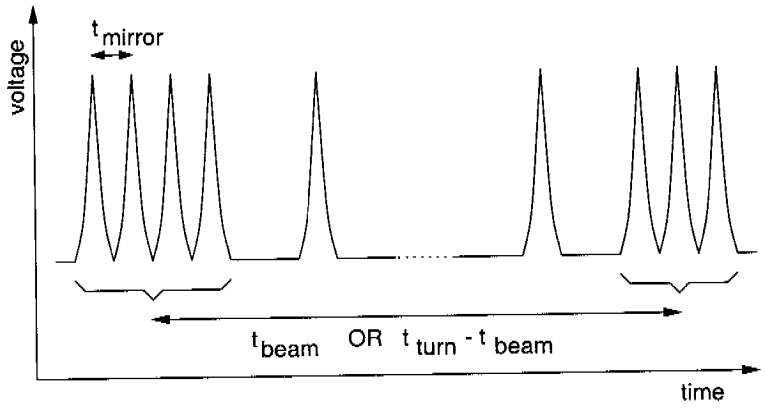


Figure 5.13: Input voltage at the parallel port as beams pass by the photo detector.

The receiver software consists of two main components, a Linux device driver which handles the parallel port interrupt, and an application level program which performs the actual location computation and lighthouse calibration. The device driver mainly consists of the parallel port interrupt handler, which is implemented using the parapin [125] parallel port programming library. Moreover, it implements a Linux special device `/proc/location`, which provides a simple interface to user-level applications. By writing simple ASCII commands to this device, a user-level program can instruct the device driver to do some action.

The supported commands are:

- `init`: Create a new lighthouse. The driver initializes data structures for a new lighthouse and autodetects the rotation speed t_{mirror} of a previously undetected lighthouse.
- `reset N`: Reset lighthouse `N` to the initial state.
- `clear`: Delete all active lighthouses.
- `rounds N M`: Instruct the driver to take into account M lighthouse turns and output the mean of the M measurements for lighthouse `N`.

By reading the `/proc/location` device, a user-level program can obtain the current status and measured angle α according to Equation 5.10 of all detected lighthouses. A sample output line looks as follows:

```
lh0: seq 1, angle 1857, pulse 4232us, rounds 1
```

indicating that lighthouse 0 has $t_{\text{mirror}} = 4232\mu$s, $\alpha = 0.01857$ (the value is scaled by 100000), has done one measurement (sequence number 1), and is taking into account 1 lighthouse turn for each measurement.

In order to measure α, the driver has to evaluate the interrupts it sees. To understand how this is done, consider Figure 5.13, which shows the input voltage

at the parallel port over time. As the first rotating laser beam passes by the photo detector, the parallel port sees a sequence of sharp pulses resulting from the fast rotating mirror. The pulses stop when the lighthouse platform has turned enough so that the photo detector isn't hit any longer by the rotating beam. After some time, the second rotating beam passes by the photo detector and again generates a sequence of fast pulses.

Recall that each pulse generates an interrupt, which results in the device driver interrupt handler being invoked. The handler then uses the system clock (which has μs resolution under Linux) to determine the point in time when the interrupt occurred.

The time interval between two successive fast pulses equals the time t_{mirror} for one rotation of the mirror. Since each lighthouse has a different t_{mirror}, this value can be used to distinguish different lighthouses. Please note that the pulse sequences can contain "holes" where the laser beam missed the photo detector due to vibrations. The driver removes all peaks separated by holes from the beginning and the end of the sequence of pulses. The time midpoint of the resulting shorter sequence of pulses without holes is assumed as the detection time of the beam (indicated by the braces in Figure 5.13).

Recall from Section 5.4.2, that we implemented a "virtual" wide beam by two rotating laser beams that form the outline of this wide beam. Therefore, the time passed between the midpoints of two successive packs is either t_{beam} or $t_{\text{turn}} - t_{\text{beam}}$. If the actual value is small (e.g., $<$ 1sec) then it is assumed to be t_{turn}. If the lighthouse has just been initialized the driver also measures $t_{\text{turn}} - t_{\text{beam}}$ in order to obtain t_{turn}. Since the latter does not change, this has to be done only once. Later on, the driver can output a new α with each round of the lighthouse.

In order to distinguish successive pulses from "holes", and holes from "beam switches", the driver knows tight lower and upper bounds for the possible values of t_{mirror} and t_{turn}. In Section 5.4.3 we mentioned the possibility that beams from different lighthouses may hit the photo receiver at the same time. If this happens the resulting time between successive pulses will fall below the lower bound for t_{mirror}, such that the driver can detect this situation instead of producing faulty results.

Using the `/proc/location` interface of the device driver, the user-level program implements lighthouse calibration and location computation. The program is more or less a straightforward implementation of the concepts developed in Section 5.4.2.

A simplified version of this software also runs on the ATMEL microcontroller of the BTnodes. This setup more closely resembles the limited capabilities of a Smart Dust node and allows us to study the potential effects of a Smart Dust node on the location system (cf. Chapter 6).

5.4.4 Evaluation

In this section we examine the compliance of the Lighthouse Location System with the region in the design space indicated in Section 5.3 and examine additional

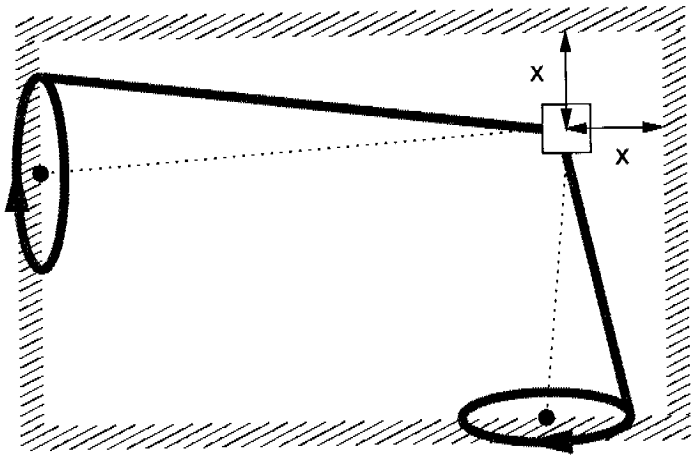


Figure 5.14: Ensuring mutually perpendicular lighthouses.

aspects of the system, namely factors that influence the precision of the location estimates, limits on the maximum distance of nodes from the base station, the effects of node mobility, and the cost of adding localization support to dust nodes. We begin with examining the calibration of the lighthouse beacon as a prerequisite for quantifying the precision of our approach.

Calibration

Calibration of the base station involves the following three steps:

- Ensuring that the lighthouses are mutually perpendicular.
- Measuring the offsets of the lighthouse centers x_0 and z_0.
- Determining $C^b, C^\beta, C^\gamma, C^\delta$ for each of the two lighthouses.

In order to ensure that the two lighthouses are mutually perpendicular, we placed the base station at the corner of a rectangular room as depicted in Figure 5.14, such that the rotation axes of the two lighthouses are at distance x from the two perpendicular walls. We disabled the motors that drive the rotating mirrors and one of the two lasers of each lighthouse. Then we adjusted the mirror so that the remaining laser beam points at the opposite wall. Due to the rotating lighthouse platforms, the laser spots draw two circles on the walls. The centers of these two circles mark the position where the lighthouse rotation axes hit the wall as depicted in Figure 5.14. Now we adjust the lighthouses on the common chassis such that the centers of these circles also have a distance x from the walls. In our measurement setup, we placed the base station at $x = 20$cm in a room with a size of about 5m by 5m.

As mentioned in Section 5.4.2, the lighthouse center offsets x_0 and z_0 from the origin of the coordinate system defined by the lighthouse rotation axes are not

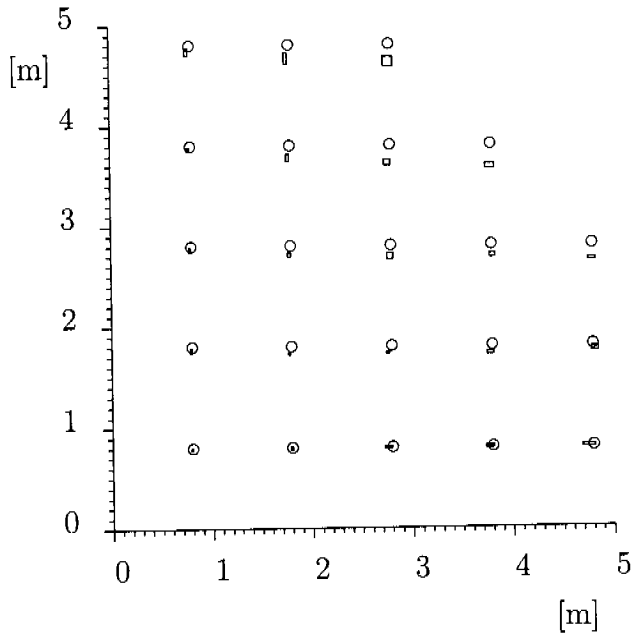


Figure 5.15: Location estimation benchmark. The ground truth locations are at the centers of the circles. The mean of the measured locations are at the centers of the boxes. The edge length of each box is twice the standard deviation in each axis.

critical for the accuracy of the system. Therefore, we measured them directly at the base station device.

In order to determine the C^* values, we placed the observer at the four locations $(x, z) \in \{(50, 50), (480, 80), (80, 480), (340, 340)\}$ (all values in centimeters) on the floor in the base station coordinate system. The respective lighthouse distance and height values are obtained from the (x, z) values as follows:

$$\begin{aligned} (d_x, h_x) &:= (z, x - x_0) \\ (d_z, h_z) &:= (x, z - z_0) \end{aligned} \tag{5.25}$$

At each of the locations, we performed ten measurements of α for each lighthouse and computed the mean value. For each of the two lighthouses we then solved Equation System 5.21 in order to obtain the C^* values.

Precision

For the precision benchmark, we placed the observer at 22 locations on the grid $(80cm + i * 100cm, 80cm + j * 100cm)$ in the base station coordinate system on the floor of the room. At each of the locations, we measured the location ten times by iteratively solving Equation System 5.19 as described in Section 5.4.2.

Figure 5.15 shows the base station coordinate system and the results of these measurements. Ground truth locations (x, z) are indicated by circles. The mean of the computed location $(\bar{x}, \bar{z})$ is at the center of the small boxes. The edge length

of each box is twice the standard deviation s_x (s_z) of the measurements in the respective axis.

Please note that we determined ground truth locations using a cheap 5m tape measure, resulting in a maximum error of about ±1cm in each axis. Also note that we did not perform outlier rejection or any other statistical "tricks" to improve the mean values or standard deviations.

The mean relative offset of the mean locations from ground truth locations (i.e., $|\bar{x} - x|/x$) is 1.1% in the x axis, and 2.8% in the z axis. The overall mean relative offset of the mean locations from ground truth locations (i.e., $|(\bar{x}, \bar{z}) - (x, z)|/|(x, z)|$ is 2.2%. The mean relative standard deviation (i.e., s_x/x) is 0.71% in the x axis and 0.74% in the z axis. The overall mean relative standard deviation (i.e., $s_{|(x,z)|}/|(x, z)|$) is 0.68%.

Sources of Imprecision

In this section we examine which factors influence the accuracy of the system. For this, we have to examine errors that can occur during the measurement of t_{beam} and t_{turn}. From a measurement point of view, the two are identical, since they are both an amount of time elapsed between two beam sightings. Therefore we will use t as a genus for the two and Δt as the absolute error of t. The following list contains possible causes for measurement errors:

- Vibrations: Due to their fast rotation, the mirrors and thus the reflected beams suffer from small vibrations, resulting in a small angle ϵ of beam spread, which is about 0.05° in our prototype. Assuming $\sin\epsilon = \epsilon$ (since $\epsilon \approx 0$), the resulting error is $\Delta t \leq t_{\text{turn}} \frac{\epsilon\sqrt{d^2+h^2}}{2\pi d}$ for an observer located at distance d from the lighthouse rotation axis and at height h over the lighthouse center.

- Lower bound on time t_{mirror} for one mirror rotation: Since we can measure elapsed time only when the rotating laser beam hits the photo detector, the accuracy of t_{beam} and t_{turn} is limited by the speed of the rotating mirrors (i.e., t_{mirror}). The resulting error is $\Delta t \leq t_{\text{mirror}}$.

- Flutter of platform rotation: The relative error in lighthouse rotation speed ρ_{lh} causes an error in t. ρ_{lh} is mainly caused by the flutter of the motor driving the lighthouse platform. The motor used in our prototype has a flutter of 0.1%. The resulting error is $\Delta t \leq t\rho_{\text{lh}}$.

- Variable delays: There is a variable time offset between the laser beam hitting the photo detector and the interrupt handler reading the clock. On the path from the photo detector to the interrupt handler are many sources of variable delay, such as hardware and interrupt latency. The actual value of this error pretty much depends on what is currently happening on the computer, but is typically small compared to the other sources of errors.

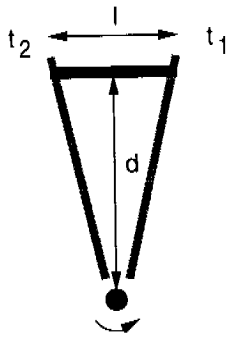


Figure 5.16: The photo detector must be hit by the laser beam at least twice.

- Clock resolution: The minimum time unit t_{min} that can be measured by the clock limits the time resolution for measurement of t. The Linux laptop we used has $t_{\text{clockres}} = 1\mu$s. On the ATMEL we used a 16-bit counter to implement a clock with $t_{\text{clockres}} = 50\mu$s. The resulting error is $\Delta t \leq t_{\text{clockres}}$.
- Clock drift: The maximum relative error ρ_{clock} in the clock rate also causes an error in t. A typical value is $\rho_{\text{clock}} = 10^{-5}$ both on Linux and the ATMEL. The resulting error is $\Delta t \leq t\rho_{\text{clock}}$.

In our prototype system, the clearly dominating errors are caused by vibrations, limited t_{mirror}, and flutter of platform rotation. The use of deflectable MEMS mirrors can both drastically reduce vibrations and t_{mirror}. The flutter of platform rotation can be reduced to about 0.01% by using electronically stabilized motors as used, for example, in turntable drives. By this, we expect a possible reduction of Δt by a factor of about 10.

Note, however, that the errors resulting from these three main sources can be modeled by a Gaussian noise source. This means that averaging over a large number of measurements helps to reduce the error.

Range

In this section we examine the maximum range, at which observers can still determine their location. This maximum range mainly depends on two issues. The first of these issues is that the photo receiver has to be hit twice by each of the rotating beams in order for the receiver to identify the lighthouse as explained in Section 5.4.3. Figure 5.16 depicts this situation. It shows a top view of a lighthouse with only *one* of the two rotating beams at *two* points in time t_1 and t_2. At t_1, the beam hits the photo detector at distance d from the lighthouse rotation axis the first time. Then, the mirror does one rotation and hits the photo detector a second time at t_2. During $t_2 - t_1$, the lighthouse platform has rotated a bit to the left. l denotes the diameter of the photo detector. Assuming a constant diameter w of the laser beam, the distance d at which the photo detector is hit at least twice is given by the following inequality:

$$d < \frac{l + w}{2\sin(\pi t_{\text{mirror}}/t_{\text{turn}})} \tag{5.26}$$

With the values of our prototype system l = 5mm, w = 3mm, t_{mirror} = 4ms, t_{turn} = 60sec we can achieve a theoretical maximum range of about 14m. This value can be improved by increasing t_{turn}, by decreasing t_{mirror}, or by defocusing the lasers a bit, such that there is a small angle of beam spread. However, there are certain limits for each of these possibilities. The angle of beam spread is limited by the sensitivity of the photo detector and the output power of the laser. t_{mirror} is limited by the possible maximum speed of the mirrors. With MEMS deflectable mirrors such as the one presented in [25], we can achieve $t_{\text{mirror}} = 1/35kHz = 30\mu s$. t_{turn} is limited by the frequency of location updates needed by the nodes and thus by the degree of node mobility (see Section 5.4.4).

The second issue that limits the maximum range of the system is the speed of the photo detector. Using commercial-off-the-shelf technology, the beam has to stay on the photo detector for about t_{photo} = 10ns in order to be detected. Depending on the minimum retention period t_{photo} of the laser beam on the photo detector, the maximum distance d is limited according to the following inequality:

$$d < \frac{l + w}{2\sin(\pi t_{\text{photo}}/t_{\text{mirror}})} \tag{5.27}$$

With the current values of our prototype t_{photo} = 200ns, t_{mirror} = 5ms, l = 4mm, w = 3mm we can achieve a theoretical maximum range of about 27m, giving us an overall range limit of 14m. Again, this value can be improved by reducing t_{mirror} and by defocusing the laser with the same limits as above.

The actually measured maximum range, at which the receiver prototype could still detect the base station is about 11 meters. However, the range can be increased by adjusting certain system parameters. A more elaborate system built using fast deflectable MEMS mirrors with values l = 1mm, w = 20mm (due to beam spread), t_{mirror} = 1ms, t_{turn} = 60sec, and t_{photo} = 10ns, for example, could achieve a theoretical maximum range of about 210m (the minimum obtained from Inequalities 5.26 and 5.27). Based on our experience, we would expect a practical maximum range of about 120-140m of a system with these parameters, which approximately equals the maximum communication range of 150m during the day for the Berkeley experiments [49].

Resources

In this section we want to examine how the presented location system fits the stringent resource restrictions of future Smart Dust Systems. These restrictions especially apply to the receiver side, i.e., the Smart Dust nodes.

The Berkeley Smart Dust prototype has already demonstrated that a photo detector similar to the one we are using for our location system is feasible. What remains to be shown is how the receiver software (i.e., the device driver and the user-level program) fit onto a Smart Dust node.

Both the processing overhead and the memory footprint of the device driver are very low, which is very important for Smart Dust. The first is true because the driver is interrupt driven, i.e., it does not do any sensor sampling or polling. Moreover, the interrupt can be used to wake up the processor from a power-saving mode. Thus, the system has to be woken up only during the short periods when a beam is hitting the photo detector. The memory footprint is very low because the driver does not have to store arrays of peak detections. Instead, for each sequence of peaks it only keeps "first peak" and "last peak" time stamps which are updated when a new interrupt occurs. The whole data structure for one lighthouse only requires about 25 bytes.

Similarly, the location-computation part of the user-level program has a very low memory footprint. It just retrieves the α values from the device driver and executes the approximation program described in Section 5.4.2. Given the relatively infrequent location updates, speed is not a problem. On computationally very limited platforms like future Smart Dust nodes, it might be necessary to revert to fixed point arithmetic and a hardware implementation of the location computation code in case the provided processing capabilities are too limited. Besides the basic arithmetic operations $(+, -, *, /)$ we need support for $\sin\alpha$ and $\sqrt{x}$ in order to solve Equation System 5.19. Note that $\sin\alpha$ is easy to approximate since the values of α obtained from Equation 5.10 are small due to $t_{\text{beam}} \ll t_{\text{turn}}$. The second order approximation $\sin\alpha \approx \alpha - \alpha^3/6$ has a maximum error of 0.1% for $|\alpha| \leq 33°$. There are also fast approximations for $y = \sqrt{x}$. One possible approach is to first approximate $1/\sqrt{x}$ by iterating $y := y(3 - xy^2)/2$ with an appropriate initial value for y. Multiplying the result by x gives an approximation for $\sqrt{x}$.

The requirements on the clock are also quite relaxed. Note that we don't need a real-time clock since we are only interested in the quotient $t_{\text{turn}}/t_{\text{beam}}$. A simple counter which ticks at a constant rate would also be sufficient. The resolution of the clock (or counter) just has to be high enough to reliably distinguish the t_{mirror} values of different lighthouses. Since the t_{mirror} values of our prototype system are 4ms and 5ms, respectively, a clock resolution of 0.5ms would be sufficient.

Please note that dust nodes don't have to be calibrated due to the following two reasons. Firstly, the two beam sightings used to measure t_{beam} and t_{turn} are identical from a measurement point of view. Any constant hardware and software delays will subtract out. Secondly, only the quotient $t_{\text{turn}}/t_{\text{beam}}$ is used for node localization, which is independent of the actual clock frequency.

Node Mobility

If nodes change their location over time, they have to update their location estimates frequently in order to avoid inaccuracies resulting from using outdated location estimates. Moreover, node movement during the measurement of parameters needed for location computation can cause inaccuracies in the estimated location.

The time t_{update} between successive location updates usually equals the time t_{turn} required for one rotation of the lighthouse. Thus, the update frequency $1/t_{\text{update}}$ can be increased by decreasing t_{turn}. However, there is an easy way to double the

update frequency when using rotating mirrors for beam generation, because the beams are reflected to both sides of the lighthouse as depicted by the dashed laser beams in Figure 5.8. Thus, we actually have two "virtual" wide beams we can use for location estimation, effectively doubling the update frequency.

If a node moves during measurement of t_{beam} (i.e., after detection of the first beam and before detection of the second beam), the obtained value of t_{beam} will be incorrect. Additional errors are caused by the node moving between measurements of t_{beam} of the three lighthouses.

There are two ways to detect and reject faulty location estimates resulting from node movement during measurement. The first compares two or more consecutive position estimates and rejects them if they differ by more than a small threshold. The second approach uses accelerometers to detect movement during measurement. Accelerometers can also be used to estimate node movement (velocity, direction) during measurements of t_{beam}. The obtained values can be used to correct t_{beam}, such that correct location estimates can also be obtained during node movement. In fact, the Smart Dust prototypes developed at Berkeley already contain such sensors.

Line-Of-Sight Requirement

As mentioned in Section 2.5.3, communication between a node and the base station requires an uninterrupted line of sight (LOS) even for "plain" Smart Dust (i.e., without using the Lighthouse Location System). Hence, the presented location system does not introduce additional restrictions with respect to LOS.

Temporary LOS obstructions can cause wrong position estimates if a dust node misses one of the laser beams. However, the probability of such errors can be reduced by comparing two or more consecutive positions estimates and rejecting them if they differ by more than a small threshold. Reflected laser beams are typically not detected by the receiver hardware, since diffuse reflection reduces the laser light intensity drastically.

Note that other localization systems based on ultrasound and radio waves provide location estimates even in the case of an obstructed line of sight. However, the resulting location estimates are typically wrong due to relying on signals reflected around the obstruction. Often it is difficult to detect such situations [42], which may result in using wrong location estimates unnoticed.

Robustness

We assume that base stations are immobile and mounted in a safe place (with respect to harmful environmental influences) due to their potential long range (see Section 5.4.4). On the other hand, dust nodes are subject to mobility and other kinds of environmental influences (e.g., LOS obstructions), which can cause faulty location estimates.

However, above we mentioned extensions to the basic system in order to detect and reject such faulty location estimates with high probability. This leaves us in a

situation, where dust nodes either obtain good position estimates or none at all.

5.5 Summary

This chapter is devoted to node localization in wireless sensor networks. We first discussed important models with respect to mobility and signal propagation. Following the framework developed in Chapter 3, we then presented important concepts used by localization algorithms. Referring to these concepts, we then presented and discussed important existing approaches to localization – both in traditional distributed systems and for sensor networks. We then identified a region in the design space that cannot be appropriately supported by these existing algorithms. In particular, we showed that tiny sensor nodes known as Smart Dust are not sufficiently supported. We presented and evaluated the Lighthouse Location System to support this region in the design space. A key feature of this approach is that sensor nodes can autonomously infer their location by observing laser light flashes emitted by a special lighthouse device. We showed that this approach is efficient since sensor nodes do not actively emit any signals. Based on a prototypical implementation, we evaluated the precision of this approach and found that an accuracy in the order of centimeters can be achieved. We also discussed potential improvements to further increase the precision.

Chapter 6

Application Experience

In this chapter we present a prototypical application in order to demonstrate the practical feasibility and usability of the approaches for time synchronization and node localization we developed in Chapters 4 and 5.

The application supports tracking the location of a mobile object – a remote-controlled toy car in our example. For this, a number of sensor nodes are deployed in the area of interest, each equipped with a sensor to detect the proximity of the mobile object. In order to estimate the current location of the tracked object, the locations of the sensor nodes in space and time must be known with respect to a common coordinate system.

In Section 6.1 we describe the general setup of the application, details are given in Sections 6.2-6.6. The tracking system is evaluated and discussed in Sections 6.7 and 6.8.

6.1 Object Tracking with Smart Dust

The purpose of our prototypical application is to track the location of real-world phenomena with a network of Smart Dust nodes. We use a remote-controlled toy car (Figure 6.1) as the tracked object. The current tracking system assumes that there is only one car. Sensor nodes are randomly deployed in the area of interest and can change their location after deployment. When they detect the presence of the car (cf. Section 6.2), they send notifications to a base station. The base station fuses these notifications (cf. Section 6.3) in order to estimate the current location of the car. A graphical user interface displays the track and allows to control various aspects of the system. The data fusion process requires that all nodes share a common reference system both in time and space, which necessitates mechanisms for node localization (cf. Section 6.4) and time synchronization (cf. Section 6.5).

Since Smart Dust hardware has not been available to us, we used BTnodes [116] to mimic the behavior of Smart Dust (cf. Section 2.5.3). Figure 6.2 shows a complete sensor node as used in our application. The base station consists of a Linux laptop computer equipped with a Bluetooth radio. In analogy to the single-

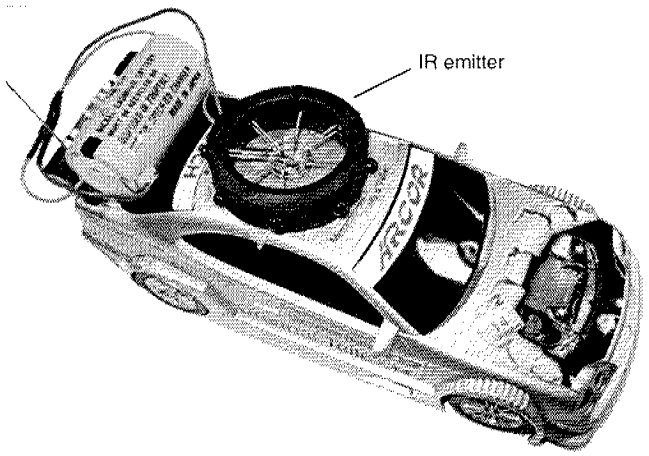


Figure 6.1: A remote-controlled toy car with an IR emitter is used as the tracked object.

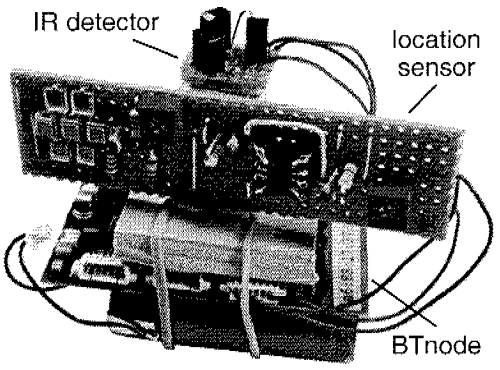


Figure 6.2: A BTnode with an IR detector and lighthouse receiver is used as a Smart Dust surrogate.

hop network topology of Smart Dust, BTnodes do not directly communicate with each other, but only with the base station. Before communication can take place, the base station has to set up a so-called Bluetooth Piconet containing no more than 7 BTnodes. To support more than 7 nodes, the base station has to periodically switch the Piconet in a round robin fashion, such that eventually every BTnode gets a chance to talk to the base station. Note the analogy to Smart Dust, where the base station has to point the (typically slightly defocused) laser beam at a group of nodes in order to enable communication with them.

6.2 Object Detection

Tracking objects with networks of sensors has been an active research topic for many years, [23, 55, 103] give a good overview of many tracking algorithms. Most of the approaches are optimized for sparse networks, where a high tracking accuracy

should be achieved despite a relatively low node density. To achieve this, many approaches make a number of assumptions about the tracked object. Methods which estimate object distance based on signal strength estimates, for example, require knowledge of the intensity of the signal emitted by the object in order to achieve good accuracy. Approaches based on measuring at different sensor nodes the difference of time of arrival of a signal emitted by the object are typically limited to sound or other signal modalities with low propagation speed. Signal modalities with high propagation speeds such as radio waves (e.g., light) would require distributed clock synchronization with an accuracy of few nanoseconds, which is typically not feasible. Other approaches require known lower and upper bounds of the velocity or the acceleration of the tracked object.

While these assumptions help to achieve good tracking accuracy, they also limit the applicability of the tracking system. In order to make our system applicable to a wide variety of objects, we tried to avoid making assumptions about the target as much as possible. In order to achieve a satisfactory tracking accuracy nevertheless, we exploit the anticipated high density of Smart Dust deployments – which is expected because of the intended small size and low cost of Smart Dust devices.

Our approach assumes that the presence of the target object can be detected with an omnidirectional sensor featuring an arbitrary but fixed sensing range r, that is, the sensor can "see" the target only if the distance to the target is lower than r. The data fusion algorithm presented in the following section needs to know an upper bound R of this sensing range. In many applications, the target cannot be instrumented for tracking purposes (e.g., a cloud of toxic gas, an oil slick, fire). The remote-controlled car we used as a sample target in our tracking system emits a characteristic acoustic signature which could be used for detection. However, this signature depends on the velocity of the car. To avoid the intricacies with detecting this variable signature, we chose in our experiment a different solution based on infrared (IR) light, leaving detection based on the car's acoustic signature as future work.

In the current version of the prototype, we equipped the car with an omnidirectional IR light emitter consisting of eight IR LEDs mounted on top of the car (Figure 6.1). Accordingly, the sensor nodes are equipped with an omnidirectional IR light detector consisting of three IR photo diodes (Figure 6.2). The used IR photo diodes include a filter to cancel out visible light. The output of the IR detector is connected to an analog-to-digital converter (ADC) of the BTnode's microcontroller. If the output value of the ADC exceeds a certain threshold, the presence of the car is assumed. Using a low-pass filter, the threshold value is adopted to slowly changing daylight, which also contains IR components. With this setup, the BTnodes can detect the car at a distance of up to approximately half a meter. When a node first detects the car, it sends a "detection notification" to the base station, containing its node ID as well as its time and location at the time of detection. When the node no longer sees the car, it sends a "loss notification" to the base station, which contains its node ID and its current time. If the node changes its location during the presence of the car, a loss notification is emitted,

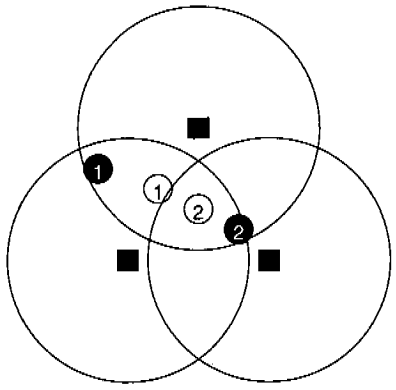


Figure 6.3: Estimating object location by centroids.

followed by a detection notification with the new node location.

6.3 Data Fusion

Following the argumentation at the begin of the Section 6.2, we try to avoid making assumptions about the target as much as possible. Therefore, the base station has to derive the current location of the tracked object solely based on detection notifications and loss notifications received from the sensor nodes.

We use an approach that estimates the car's location at time t by the centroid of the locations of the sensor nodes that "see" the car at time t. The centroid of a set of N locations $\{l_i = (x_i, y_i, z_i)\}$ is defined as $\hat{l} := \frac{1}{N}\sum l_i = \left(\frac{1}{N}\sum x_i, \frac{1}{N}\sum y_i, \frac{1}{N}\sum z_i\right)$. Consider Figure 6.3 for an example. Depicted are three sensor nodes (black squares) with their respective sensing ranges, and two car locations (black circles). The hollow circles indicate the respective estimated locations (i.e., centroids).

Figure 6.4 illustrates an algorithm to calculate the car location estimates given the detection and loss notifications received from the sensor nodes as described in Section 6.2. The figure shows sensor nodes 1 and 2, their respective sensing ranges, and the trajectory of the car (dotted arrow). When the car enters the sensing range of node i, a detection notification d_i is emitted, containing time $d_i.t$ and location $d_i.l$ of node i at the time of detection. Accordingly, node i emits a loss notification l_i when the car leaves the sensing range. In a first step, all notifications are sorted by increasing time stamps $d_i.t$ ($l_i.t$) as depicted on the time axis in the lower half of Figure 6.4. In a second step, we iterate over these sorted notifications from left to right, recording the active nodes (those that currently see the car) in a set S. If we come across a loss notification l_i, we remove i from S. If we come across a detection message d_i, we add i to S. Additionally, we remove all nodes j from S, whose sensing ranges do not overlap with the detection range of node i, that is, for which $|d_i.l - d_j.l| > 2R$ holds. This is necessary to compensate for missed loss notifications, which would otherwise permanently affect the accuracy of the tracking system by not removing the respective entries from S. A missing enter notification will lead to a temporarily decreased tracking accuracy, but will not

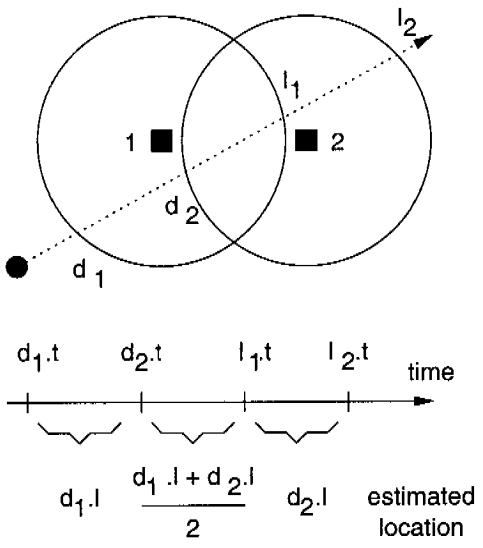


Figure 6.4: Data fusion algorithm.

otherwise permanently affect the system.

The location of the car during the time interval starting at the currently considered notification and ending at the next notification is estimated by the centroid $\hat{S}$ of the locations of the nodes in S (i.e., $d_1.l$ during $[d_1.t, d_2.t)$, $(d_1.l + d_2.l)/2$ during $[d_2.t, l_2.t)$, and $d_2.l$ during $[l_1.t, l_2.t)$).

The localization accuracy of a similar centroid-based algorithm was examined in [17] in a different context under the assumption that nodes are located on a regular grid. We can interpret their results for our setting as follows. The localization accuracy depends on the sensing range r of the nodes (about 50 cm in our case) and the distance d between adjacent nodes. For $r/d = 2$ (i.e., $d \approx 25$ cm in our case) the average and maximum localization errors are $0.2d$ (i.e., 5 cm) and $0.5d$ (i.e., 12.5 cm), respectively. In general, larger r/d values yield better accuracy. Therefore, the accuracy can be improved by increasing the node deployment density, since that reduces d while keeping r constant.

6.4 Node Localization

In order to derive the location of the tracked car from proximity detections as described in Section 6.2, the locations of the sensor nodes have to be estimated. We used the Lighthouse Location System for this purpose. The receiver hardware described in Section 5.4.3 was connected to a digital input pin of the ATMEL microcontroller, such that every light flash triggers an interrupt. A simplified version of the receiver software (omitting support for calibration) has been ported to the ATMEL.

6.5 Time Synchronization

The car location estimation described in Section 6.3 assumes that the timestamps contained in notification messages refer to a common physical time scale. We used Time-Stamp Synchronization for this purpose. We implemented a slight variation in order to relief the base station from keeping state information for every node (cf. Section 4.4.4). This can be achieved by sending this state information to the node as part of the acknowledgment message. A node must then include this state information in the next event notification to make it again available to the base station. This way, every node is responsible for storing its own state.

6.6 Message Ordering

The data fusion algorithm described in Section 6.3 requires sorting notifications by their time stamps. The time transformation approach described in Section 6.5 enables us to compare and sort time stamps originating from different nodes. However, we still have to ensure that a notification message is not processed by the data fusion algorithm until all earlier notifications have arrived at the base station. This is of particular importance for Smart Dust, since messages are subject to long and variable delays as described in Section 2.5.3.

One particularly attractive approach to message ordering is based on the assumption that there is a known maximum network latency D. Delaying the evaluation of inbound messages for D will ensure that out-of-order messages will arrive during this artificial delay and can be ordered correctly using their time stamps. That is, message ordering can be achieved without any additional message exchanges. The literature discusses a number of variants of this basic approach [59, 70, 94].

However, there is one major drawback of this approach: the assumption of a bounded and known maximum network latency. Since communication requires that the base station points its laser at a node, Smart Dust suffers from long and variable network delays. Using a value for D which is lower than the actual network latency results in messages being delivered out of order. Using a large value for D results in long artificial delays, which unnecessarily decreases the performance of the tracking system.

We therefore introduce a so-called adaptive delaying technique that measures the actual network delay and adapts D accordingly. Doing so, it is possible that the estimated D is too small and messages would be delivered out of order. Our algorithm detects such late messages and deletes them (i.e., does not deliver them to the application at all). Recall that the data fusion algorithm presented in Section 6.3 was specifically designed to tolerate missing detection and loss notifications. Hence, deleting a message only results in a less accurate track, since the affected Smart Dust node simply does not contribute to the estimation of the target location. We argue that this slight decrease of accuracy is acceptable since deleting a message is a rare event, which only occurs at startup or when the maximum

network latency increases during operation (i.e., when the value of D is lower than the actual maximum network latency). The expected high density of Smart Dust deployments can also compensate this decrease of accuracy. Additionally, our algorithm includes a parameter which can be tuned to trade off tracking latency for tracking accuracy.

Specifically, the adaptive delaying algorithm executed in the base station maintains a variable D holding the current estimate of the maximum delay, a variable t_{latest} holding the time stamp of the latest message delivered to the application, and a queue which stores messages ordered by increasing time stamps. Initially, D is set to some estimate of the maximum network latency, t_{latest} is set to the current time t_{now} in the base station, and the queue is empty.

Upon arrival of a new notification n with time stamp (interval) $n.t$, the actual message delay $d := t_{\text{now}} - n.t^l$ is calculated[1]. D is then set to the maximum of D and $c \cdot d$. The constant factor c influences the chance of having to delete an out-of-order message and can thus be tuned to trade off tracking latency for tracking accuracy. We use $c = 1.2$ in our prototype. Now we check if $t_{\text{latest}} < n.t^l$ holds, in which case n can still be delivered in order. If so, n is inserted into the queue at the right position according to $n.t$. Otherwise, n is deleted.

The first element n_0 of the queue (i.e., the one with the smallest time stamp) is removed from the queue as soon as the base station's clock (t_{now}) shows a value greater than $n_0.t^r + D$. Now t_{latest} is set to $n_0.t^r$ and n_0 is delivered to the data fusion algorithm.

6.7 Evaluation

In order to assess the precision of the proposed tracking system, we performed a set of measurements. Figure 6.5 shows the setup of our measurements. The lighthouse beacon ("LH" in the picture) was placed in the upper left corner and defines the origin (0,0) of a 2D coordinate system. Six sensor nodes (numbered rectangles in the figure) were placed in an area of about one square meter. The car moved then through the sensor field from right to left on a straight line. Location estimates were obtained at 12 positions of the car (indicated by the black squares in the picture). We performed the whole experiment 10 times and calculated averages. The sensor nodes as well as the car locations are annotated with coordinates $(x \pm e_x, y \pm e_y)$, where (x, y) are the ground truth positions in centimeters obtained by a tape measure. $\pm e_x$ and $\pm e_y$ indicate the average errors of the output of the tracking system relative to the ground truth position on the x and y axis, respectively. The *average* error of the sensor node location estimates is $\bar{e_x} = 4.16\,\text{cm}$ and $\bar{e_y} = 1.83\,\text{cm}$. We attribute the larger $\bar{e_x}$ value to mechanical problems with one of the lighthouses. The *average* error of the car location estimates is $\bar{e_x} = 12.5\,\text{cm}$ and $\bar{e_y} = 3.5\,\text{cm}$. The *maximum* error of the sensor node location estimates is $\hat{e_x} = 5\,\text{cm}$ and $\hat{e_y} = 2\,\text{cm}$. The *maximum* error of the car location estimates is $\hat{e_x} = 28\,\text{cm}$

[1] t^r (t^l) refers to the right (left) end of the time interval t.

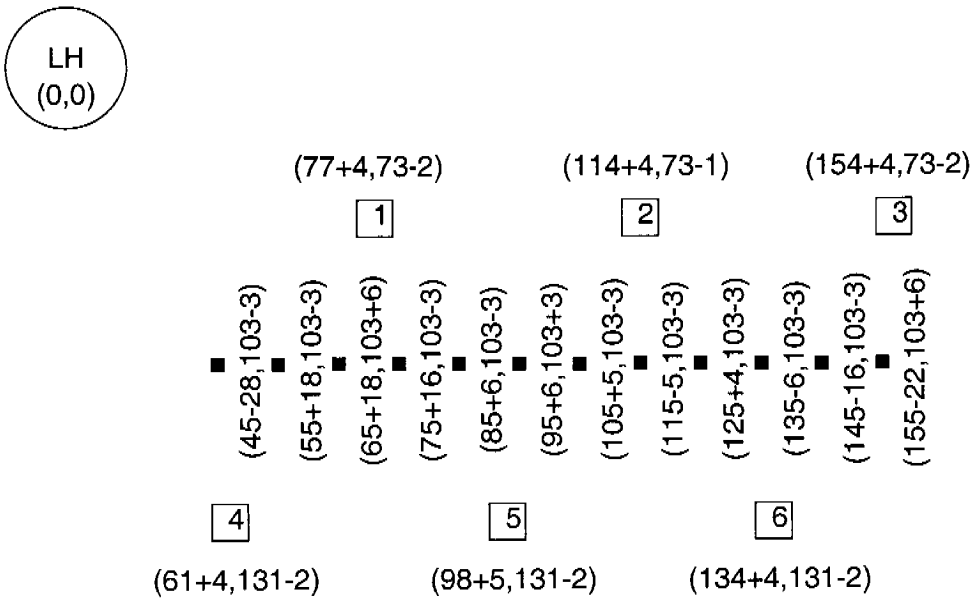


Figure 6.5: Measurement setup (not drawn to scale).

and $\hat{e_y} = 6\,\text{cm}$. The difference between the values for the x and y axis is due to the asymmetry of the node arrangement.

The tracking latency is defined as the delay after which the state of the real world is reflected by the output of the tracking system. This delay depends on the following additive components: (1) the sampling interval of the sensors, (2) processing delays in the sensor nodes, (3) the network latency, (4) delays caused by the message ordering algorithm, and (5) delays caused by the algorithm used to compute the target location estimate. The minimum value of (1) heavily depends on the sensor and processor hardware. In our implementation, (1) is limited to about 0.1 ms by the analog-to-digital converter. Components (2) and (4) are small in our system due to the simplicity of the used algorithms.

To evaluate the tracking latency of our system, we measured the sum of (2), (3), (4), and (5) by calculating the age of each notification after it has been processed by the location estimation algorithm. During the course of our experiment, the average age was 56 ms. We also monitored the value of D used by the message ordering algorithm. We used an initial guess of $D = 20\,\text{ms}$. At the beginning of the experiment, this value was quickly adapted to 52 ms. Recall from Section 6.6 that messages may be dropped by the ordering algorithm if the value used for D is lower than the actual maximum network latency. Surprisingly, during our experiments not a single message was dropped. This is due to the fact that the time between arrival of successive notifications at the base station was always greater than the network latency in our experiment. However, this is typically not the case for a real deployment, where the network latency can be significantly larger and where many densely deployed nodes may detect the target almost concurrently and generate according notifications in short intervals.

Figure 6.6 shows the above measurement setup as depicted by the graphical user

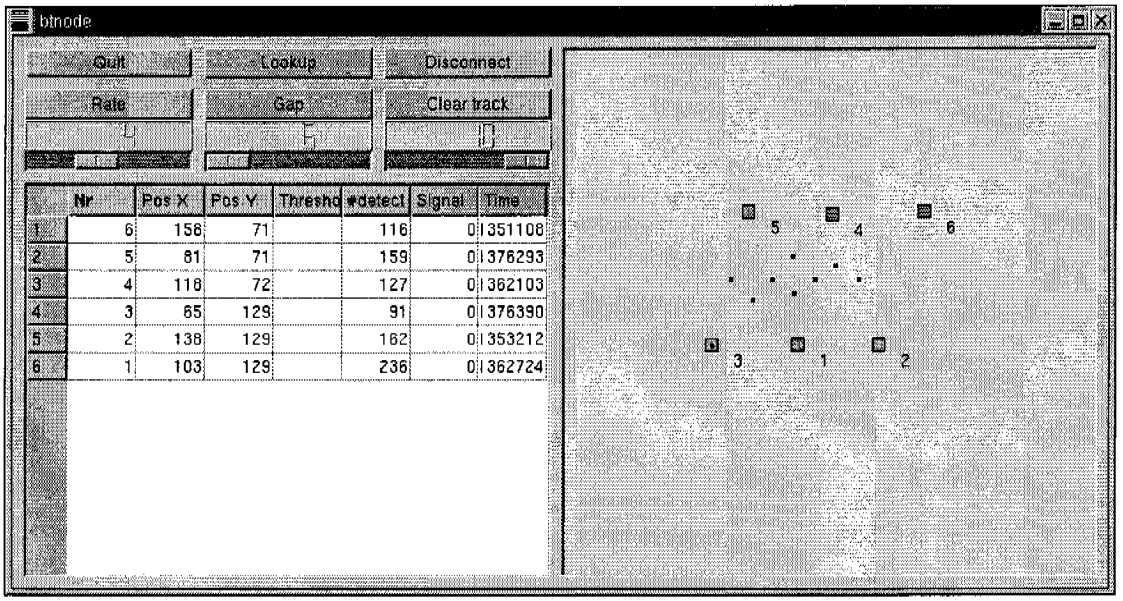


Figure 6.6: Measurement setup as shown by the graphical user interface.

interface. In the top left, a number of controls are shown to lookup sensor nodes ("Lookup"), to disconnect from the sensor nodes ("Disconnect"), to adjust the frequency of sensor readout ("Rate"), to control the detection threshold ("Gap"), and to clear the displayed track ("Clear track"). The table below the controls contains one line for each sensor node, showing x and y position, the current detection threshold, number of detections, the currently detected signal strength, and the time of the last detection. On the right, a display of the tracking area is shown, depicting the sensor nodes (larger rectangles) and some of the location estimates of the car (smaller squares) moving from right to left.

6.8 Discussion

In the presented tracking system, we make use of the scalability of Time-Stamp Synchronization and Lighthouse Location for a tracking application. This is achieved in part by strictly avoiding inter-node communication. Additionally, the algorithms employed in the base station are designed to perform independent of the actual number of nodes in the network. Instead, the overhead of the base station algorithms depends on the number of *active* nodes – those that currently "see" the tracked target. Also, the base station only has to store state information for active nodes.

Despite some similarities in the communication scheme (both true Smart Dust and our prototype use a single-hop network topology), there is one important difference between our prototype and a system based on true Smart Dust. While in our system nodes can send messages to the BST at any time, communication with a Smart Dust node requires that the BST points its laser beam at that particular

node. Even though a slightly defocused laser beam would enable the base station to communicate with many Smart Dust nodes at a time, the laser has to sweep over the deployment area to give each node a chance to talk to the base station. We deal with the resulting long and variable network delays by proposing a message ordering technique which adapts to the actual maximum network latency.

The data fusion algorithm used in our system might seem somewhat simplistic compared to many approaches described in the literature. However, it achieves a reasonable accuracy while only making a minimum of assumptions about the tracked object. The loss of precision can be compensated by increasing the node density – which is possible due to the expected small size and low cost of Smart Dust nodes.

The tracking system has been designed to tolerate node failures, since these are likely to happen. Messages lost due to node failures will only affect the accuracy of the estimated track. Again, this can be compensated by a higher node deployment density.

6.9 Summary

This chapter shows the practical feasibility of our approaches for time synchronization and node localization using a concrete application. In particular, we showed how Time-Stamp Synchronization and Lighthouse Localization can be used to implement a scalable tracking system based on Smart Dust. We also presented an approach to implement message ordering for networks with long, variable delays such as Smart Dust. We also evaluated the precision of our prototype and found that it provides a precision in the order of tens of centimeters.

Chapter 7

Conclusions and Future Work

In this final chapter, we summarize the contributions of our work and discuss some limitations of our approaches. We also sketch potential future work, both with respect to the proposed algorithms and in the broader context of our work.

7.1 Contributions

We prosed a *design space* to replace the narrow definition of wireless sensor networks that is currently assumed by most research papers. The selection of the dimensions of the design space was based on an extensive study of existing prototypical applications. It was shown that these applications do indeed cover different points in the design space. The identified dimensions of the design space *deployment*, *mobility*, *resources*, *heterogeneity*, *communication modality*, *infrastructure*, *network topology*, *coverage*, *connectivity*, *network size*, and *QoS requirements* typically have a significant impact on the design of specific solutions. We examined this impact on algorithms for time synchronization and node localization. In particular, we identified important regions in the design space which are not covered by existing solutions.

With respect to time synchronization, we showed that existing algorithms do not appropriately support intermittent and sporadic network connectivity. We proposed *Time-Stamp Synchronization* to overcome this lack. In contrast to existing schemes, this approach does not synchronize clocks. Rather, each unsynchronized clock defines its own local time scale. Time-Stamp Synchronization transforms time stamps between these time scales as they are passed from node to node, also across temporary network partitions. Moreover, synchronization can be performed on demand, where and when needed. Our approach uses intervals to represent time stamps. Therefore, reasoning about time stamps does not suffer from problems due to synchronization errors. Also, our approach is efficient because it can often piggyback time information on existing network traffic.

With respect to node localization, we showed that existing algorithms do not appropriately support tiny sensor nodes known as *Smart Dust*. We proposed the *Lighthouse Location System* to overcome this lack. This approach is based on a

new cylindrical lateration scheme, which allows Smart Dust nodes to autonomously estimate their locations with high precision using only a single external beacon device that can be integrated with an existing base station. Since nodes do not require the support of other nodes for localization, this approach can scale to very dense networks, and localization can be performed on demand where and when needed.

We demonstrated the practical feasibility of the proposed solutions by implementations that have also been used in a common prototypical application that supports tracking of mobile objects with a sensor network.

As one further contribution, we developed a common framework for discussing time synchronization and node localization. In particular, we showed that many existing distributed algorithms for synchronization and localization share five common structural elements: *bootstrapping*, *obtaining constraints*, *combining constraints*, *selecting constraints*, and *maintaining localization over time*.

The major contributions of this thesis have also been published, most notably in [34, 80, 82, 83, 84, 85, 86].

7.2 Limitations

The approaches to time synchronization and localization we proposed are tailored to specific regions in the design space as discussed in Sections 4.3 and 5.3. Naturally, our approaches are less suited for other regions in the design space, resulting in a number of limitations.

7.2.1 Time-Stamp Synchronization

Traditional time-synchronization schemes proactively synchronize clocks using a separate protocol *before* the application may request synchronized time. Hence, these protocols can spend significant effort into achieving good precision. For example, they may perform multiple measurements to obtain many constraints. In contrast, our scheme is designed to be performed on demand at the instant when the application requests synchronized time. To make such an approach feasible, the overhead of synchronization must be kept to a minimum. For example, performing multiple measurements is disadvantageous here, effectively limiting the precision of time-stamp synchronization.

Time-Stamp Synchronization provides synchronization only for selected time instants. This is not sufficient for applications that require continuous synchronization over longer periods of time (e.g., performing periodic actions at a certain rate).

7.2.2 Lighthouse Location System

Localization with our approach requires a free line of sight between the lighthouse beacon and sensor nodes, which limits the applicability of this approach. Note,

however, that the basic communication scheme of Smart Dust does also impose this restriction. Hence, our localization system does not introduce an additional constraint here.

The geographical scale of networks that can be supported with our approach is limited by the range of the lighthouse beacon. The Lighthouse Location System uses range measurements to multiple lighthouses as a basic element. To obtain correct results, the sensor node must not move during the measurement. Hence, this approach cannot directly support nodes which are in constant movement unless additional sensors (e.g., acceleration) are used to compensate for movements during measurement.

7.3 Future Work

There are a number of potential improvements with respect to our algorithms for time synchronization and localization. However, we also discuss future work in the broader context of our work, in particular with respect to the design space.

7.3.1 Time-Stamp Synchronization

For some applications it might be reasonable to strive for improved precision at the cost of increased runtime overhead. Some directions which might be worth further investigation have been sketched in Section 4.4.9. *Probability distributions* instead of pure intervals would allow probabilistic reasoning about overlapping time intervals. Such distributions could perhaps be constructed by taking into account past message exchanges between a pair of nodes. A time-stamp *history* could be included in each message to improve reasoning with time stamps that passed through a common node. However, this may result in significantly increased messages size.

7.3.2 Lighthouse Location System

It would be interesting to examine the use of deflectable MEMS mirrors or steerable lasers instead of rotating mirrors, since this could not only significantly improve the precision of localization, but would also lead to a simplified and more robust lighthouse hardware. On the node side, an implementation on true Smart Dust would gain significant insights into the feasibility of our approach on this hardware platform.

For networks with a large geographical extension, a single lighthouse beacon may not be sufficient to cover the whole network. Here, multiple lighthouse beacons could be used. Future work would include an examination how a consistent coordinate system could be established across a large network with multiple beacons. A natural approach would be to use clustering. Each lighthouse beacon would define a cluster, such that all nodes in the cluster can receive the respective beacon. Each cluster uses the coordinate system defined by its lighthouse beacon. Nodes

that are contained in multiple clusters could compute coordinate transformations between adjacent clusters.

Another issue for future work is better support for mobility. As a first step, movement detectors could be used to detect faulty measurements, caused by a node moving during measurement. A second step would include movement compensation, so that localization becomes also possible if a node is in constant travel. Such an approach would require some form of dead reckoning (e.g., by means of acceleration sensors) to compensate the error in the sweep time measurement caused by node movement.

7.3.3 Service Interfaces

Time synchronization and node localization can be considered as two basic services that each provide one major function at its interface to the application: *What is the current time/location?* However, as we have illustrated in this thesis, different points in the design space may require vastly different approaches to provide this basic function in an adequate way. Also, concrete application requirements (e.g., precision, scope, lifetime, mobility model) may necessitate different approaches for time synchronization and node localization.

Despite these different requirements, modular programming could be greatly enhanced if time synchronization and node localization services could be considered basic building blocks that provide a common interface across the design space. This would, however, require the inclusion of methods into the interface that allow a specification of the concrete requirements on the services, which might also change during the lifetime of the application. This raises the important question of how such an extended interface should look like. Broadly speaking, such an interface should include methods to specify the exact point in the design space and additional application requirements.

7.3.4 Service Selection and Adaptation

Different points in the design space and different application requirements typically necessitate different solutions for time synchronization and node localization. In some cases, a single algorithm may provide parameters for adaptation to different requirements. In any case, an application developer is faced with the problem of selecting an appropriate solution and/or appropriate parameters. If these requirements change during the lifetime of an application, this choice may have to be updated every now and then.

To relief application developers from these issues, frameworks and tools should provide support for service selection and adaptation. Such system support is facilitated by a common service interface as discussed in the previous section. By this, different service implementations become interchangeable. The interface elements that allow a user to specify application requirements and a point in the design space could then be used by the system to automatically select an appropriate service

implementation.

7.3.5 Calibration

We can consider localization in both time and space as *sensor calibration* problems. Generally speaking, a sensor is a device that takes a certain physical quantity as input and produces a variable electrical signal as output that is usually converted to a digital number using an analog-to-digital converter. Calibration then consists of enforcing a certain mapping of the observed physical quantity to the sensor output. For example, if a sensor is exposed to a temperature of T°C, then the sensor should output T (in some reasonable digital encoding). Localization in space can then be considered as the task of calibrating a location sensor to a given coordinate system. Localization in time can likewise be considered as the task of calibrating a time sensor (e.g., a hardware clock) to a given coordinate system.

It would be worthwhile to examine the use of known techniques from time synchronization and node localization in the more general context of calibration. It particular, it is quite likely that many of the observations in Chapter 3 could be generalized to calibration. For example, the classification in Section 3.2 can be directly transferred to calibration. It is also conceivable that distributed calibration algorithms could consist of structural elements similar to those we identified in Section 3.3.

7.4 Concluding Remarks

In this thesis we showed that applications of wireless sensor networks cannot be characterized by a single, narrow definition, and we proposed a multi-dimensional design space of Wireless Sensor Networks. We examined the implications of this design space on methods for time synchronization and node localization and found that existing techniques do not appropriately support important regions in the design space. We proposed novel approaches to fill this gap, yielding a more comprehensive understanding and solution space of localization and time synchronization in sensor networks.

Bibliography

[1] H. Abelson et al. Amorphous Computing. *CACM*, 43(5):74–82, March 2000.

[2] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[3] S. Antifakos, F. Michahelles, and B. Schiele. Proactive Instructions for Furniture Assembly. In *Proc. Ubicomp 2002*, Gothenburg, Sweden, September 2002.

[4] A. Aoe, A. Bell, and S. Rodriguez. Multi-Modal Mobile Security Sensing. Technical report, University of California, San Diego, 2003.

[5] P. Ashton. Algorithms for off-line clock synchronization. Technical Report TR COSC 12/95, Department of Computer Science, University of Canterbury, December 1995.

[6] H. Baldus, K. Klabunde, and G. Muesch. Reliable Set-Up of Medical Body-Sensor Networks. In *Proc. EWSN 2004*, Berlin, Germany, January 2004.

[7] S. Bapat. *Object–Oriented Networks, Models for Architecture, Operations, and Management.* Prentice–Hall International, 1994.

[8] P. Basu and T. D. C. Little. Networked Parking Spaces: Architecture and Applications. In *VTC Fall*, Vancouver, Canada, September 2002.

[9] M. Bauer, L. Jendoubi, and O. Siemoneit. Smart Factory Mobile Computing in Production Environments. In *WAMES 2004*, Boston, USA, June 2004.

[10] P. H. Bauer, M. Sichitiu, R. S. H. Istepanian, and K. Premaratne. The Mobile Patient: Wireless Distributed Sensor Networks for Patient Monitoring and Care. In *ITAB-ITIS 2000*, Arlington, USA, November 2000.

[11] R. Beckwith, D. Teibel, and P. Bowen. Pervasive Computing and Proactive Agriculture. In *Adjunct Proc. PERVASIVE 2004*, Vienna, Austria, April 2004.

[12] M. Beigl, H.W. Gellersen, and A. Schmidt. MediaCups: Experience with Design and Use of Computer-Augmented Everyday Objects. *Computer Networks, Special Issue on Pervasive Computing*, 25(4):401–409, March 2001.

[13] M. Beigl, A. Krohn, T. Zimmer, C. Decker, and P. Robinson. AwareCon: Situation Aware Context Communication. In *Ubicomp 2003*, Seattle, USA, October 2003.

[14] J. Beutel, O. Kasten, F. Mattern, K. Römer, L. Thiele, and F. Siegemund. Prototyping Sensor Network Applications with BTnodes. In *EWSN 2004*, Berlin, Germany, January 2004.

[15] P. Blum, L. Meier, and L. Thiele. Improved interval-based clock synchronization in sensor networks. In *Proceedings of the Third International Symposium on Information Processing in Sensor Networks (IPSN '04)*, 2004.

[16] BP. Smart Surrogates. *Frontiers - The BP Magazine of Technology and Innovation*, (9), 2004. Online at subsites.bp.com/company_overview/technology/frontiers/fr09apr04/-fr09sensors.asp.

[17] N. Bulusu, J. Heideman, and D. Estrin. GPS-less Low Cost Outdoor Localization for Very Small Devices. *IEEE Personal Communications*, 7(5):28–34, October 2000.

[18] Z. Butler, P. Corke, R. Peterson, and D. Rus. Networked Cows: Virtual Fences for Controlling Cows. In *WAMES 2004*, Boston, USA, June 2004.

[19] V. Bychkovskiy, S. Megerian, D. Estrin, and M. Potkonjak. A Collaborative Approach to In-Place Sensor Calibration. In *IPSN*, Palo Alto, USA, April 2003.

[20] S. Capkun, M. Hamdi, and J. P. Hubaux. GPS-free Positioning in Mobile Ad Hoc Networks. In *34th International Conference on System Sciences*, Hawaii, January 2001.

[21] S. Capkun and J. P. Hubaux. Secure positioning of wireless devices with application to sensor networks. In *Infocom*, Miami, USA, March 2005.

[22] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat Monitoring: Application Driver for Wireless Communications Technology. In *2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, San Jose, Costa Rica, April 2001.

[23] J. C. Chen, K. Yao, and R. E. Hudson. Source Localization and Beamforming. *IEEE Signal Processing Magazine*, 19(2):30–39, 2002.

[24] S. Coleri, S. Y. Cheung, and P. Varaiya. Sensor Networks for Monitoring Traffic. In *Allerton Conference on Communication, Control, and Computing*, Urbana-Champaign, USA, October 2004.

[25] R. Conant, J. Nee, K. Lau, and R. Muller. A Fast Flat Scanning Micromirror. In *2000 Solid-State Sensor and Actuator Workshop*, Hilton Head, USA, June 2000.

[26] H. Dai and R. Han. Tsync: A lightweight bidirectional time synchronization service for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(1):125–139, January 2004.

[27] K. A. Delin. The Sensor Web: A Macro-Instrument for Coordinated Sensing. *Sensors*, 2(1):270–285, 2002.

[28] K. A. Delin, R. P. Harvey, N. A. Chabot, S. P. Jackson, M. Adams, D. W. Johnson, and J. T. Britton. Sensor Web in Antarctica: Developing an Intelligent, Autonomous Platform for Locating Biological Flourishes in Cryogenic Environments. In *Lunar and Planetary Science Conference*, League City, USA, March 2003.

[29] A. Dey. *Providing Architectural Support for Building Context-Aware Applications.* PhD thesis, College of Computing, Georgia Tech, 2000.

[30] L. Doherty, K. S. J. Pister, and L. E. Ghaoui. Convex Position Estimation in Wireless Sensor Networks. In *Infocom 2001*, Anchorage, Alaska, April 2001.

[31] A. Duda, G. Harrus, Y. Haddad, and G. Bernard. Estimating global time in distributed systems. In *7th International Conference on Distributed Computing Systems (ICDCS'87)*, Berlin, Germany, September 1987. IEEE.

[32] N. Eagle and A. S. Pentland. Social Network Computing. In *Ubicomp 2003*, Seattle, USA, September 2003.

[33] J. Elson, L. Girod, and D. Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *OSDI 2002*, Boston, USA, December 2002.

[34] J. Elson and K. Römer. Wireless Sensor Networks: A New Regime for Time Synchronization. *ACM SIGCOMM Computer Communication Review (CCR)*, 33(1):149–154, January 2003.

[35] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59–69, January 2002.

[36] L. Evers, S. Dulman, and P. Havinga. A distributed precision based localization algorithm for ad-hoc networks. In *PERVASIVE 2004*, pages 269–286, Vienna, Austria, April 2004.

[37] M. Fedak, P. Lovell, B. McConnell, and C. Hunter. Overcoming the Constraints of Long Range Telemetry from Animals: Getting More Useful Data

from Smaller Packages. *Integrative and Comparative Biology*, 42(1):3–10, 2002.

[38] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[39] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Technical Report CSD-TR 02-0013, UCLA, February 2002.

[40] D. Ganesan, S. Ratnasamy, H. Wang, and D. Estrin. Coping with Irregular Spatio-Temporal Sampling in Sensor Networks. *SIGCOMM Computer Communication Review*, 34(1):125–130, 2004.

[41] L. Girod, V. Bychkovskiy, J. Elson, and D. Estrin. Locating Tiny Sensors in Time and Space: A Case Study. In *International Conference on Computer Design (ICCD) 2002*, Freiburg, Germany, September 2002.

[42] L. Girod and D. Estrin. Robust range estimation using acoustic and multimodal sensing. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS) 2001*, Maui, Hawaii, October 2001.

[43] M. Hazas and A. Ward. A Novel Broadband Ultrasonic Location System. In *Ubicomp 2002*, Goteborg, Sweden, September 2002.

[44] T. He, C. Huang, B. M. Blum, J. A. Stankovic, and T. Abdelzaher. Range-free localization schemes for large scale sensor networks. In *Mobicom*, San Diego, USA, September 2003.

[45] T. He, S. Krishnamurthy, J. A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, and L. Gu. Energy-Efficient Surveillance System Using Wireless Sensor Networks. In *Mobisys*, Boston, USA, June 2004.

[46] J. Ho, B. Ramasamy, and R. Steccy. Portable Air Vent Calibration System. Technical report, University of California, San Diego, 2003.

[47] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning System: Theory and Practice, 4th Edition.* Springer-Verlag, 1997.

[48] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *Proc. ASPLOS X*, San Jose, USA, October 2002.

[49] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Emerging Challenges: Mobile Networking for Smart Dust. *Journal of Communications and Networks*, 2(3):188–196, September 2000.

[50] C. Kappler and G. Riegel. A Real-World, Simple Wireless Sensor Network for Monitoring Electrical Energy Consumption. In *Proc. EWSN 2004*, Berlin, Germany, January 2004.

[51] V. A. Kottapalli, A. S. Kiremidjian, J. P. Lynch, E. Carryer, T. W. Kenny, K. H. Law, and Y. Lei. Two-Tiered Wireless Sensor Network Architecture for Structural Health Monitoring. In *SPIE Symposium on Smart Structures and Materials*, San Diego, USA, March 2000.

[52] M. LaFiandra, K. Ragay, and W. Mulyadi. Fast Information Response Environment (FIRENet). Technical report, University of California, San Diego, 2003.

[53] A. Lamarca, W. Brunette, D. Koizumi, M. Lease, S. Sigurdsson, K. Sikorski, D. Fox, and G. Borriello. PlantCare: An Investigation in Practical Ubiquitous Systems. In *Ubicomp*, Gothenburg, Sweden, September 2002.

[54] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(4):558–565, July 1978.

[55] D. Li, K. D. Wong, Y. H. Hu, and A. M. Sayeed. Detection, Classification, and Tracking of Targets. *IEEE Signal Processing Magazine*, 19(2):17–29, 2002.

[56] Q. Li and D. Rus. Global clock synchronization in sensor networks. In *IEEE InfoCom*, Hong Kong, China, March 2004.

[57] B. Liskov. Practical Uses of Synchronized Clocks in Distributed Systems. *Distributed Computing*, 6(4):211–219, 1993.

[58] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *WSNA*, Atlanta, USA, September 2002.

[59] M. Mansouri-Samani and M. Sloman. GEM – A Generalised Event Monitoring Language for Distributed Systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4(25), February 1997.

[60] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi. The flooding time synchronization protocol. In *SenSys*, Baltimore, USA, November 2004.

[61] I. W. Marshall, C. Roadknight, I. Wokoma, and L. Sacks. Self-Organizing Sensor Networks. In *UbiNet 2003*, London, UK, September 2003.

[62] K. Martinez, R. Ong, J. K. Hart, and J. Stefanov. GLACSWEB: A Sensor Web for Glaciers. In *Adjunct Proc. EWSN 2004*, Berlin, Germany, January 2004.

[63] K. Marzullo and S. Owicki. Maintaining the time in a distributed system. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 295–305. ACM Press, 1983.

[64] F. Mattern. Virtual Time and Global States in Distributed Systems. In *Workshop on Parallel and Distributed Algorithms*, Chateau de Bonas, October 1988.

[65] L. Meier, P. Blum, and L. Thiele. Internal synchronization of drift-constraint clocks in ad-hoc sensor networks. In *Proceedings of the Fifth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2004)*, 2004.

[66] W. M. Meriall, F. Newberg, K. Sohrabi, W. Kaiser, and G. Pottie. Collaborative Networking Requirements for Unattended Ground Sensor Systems. In *Proc. IEEE Aerospace Conference*, March 2003.

[67] F. Michahelles, P. Matter, A. Schmidt, and B. Schiele. Applying Wearable Sensors to Avalanche Rescue. *Computers and Graphics*, 27(6):839–847, 2003.

[68] D. L. Mills. Improved algorithms for synchronizing computer network clocks. In *Conference on Communication Architectures (ACM SIGCOMM'94)*, London, UK, August 1994. ACM.

[69] A. Nasipuri and K. Li. A directionality based location discovery scheme for wireless sensor networks. In *WSNA*, Atlanta, USA, September 2002.

[70] G. J. Nelson. *Context-Aware and Location Systems.* PhD thesis, University of Cambridge, 1998.

[71] D. Niculescu and B. Nath. Ad hoc positioning system (aps). In *GLOBECOM*, San Antonio, USA, November 2001.

[72] Parliamentary Office of Science and Technology. Intelligent Transport. *Postnote*, 2002(187), 2002.

[73] C. Plessl, R. Enzler, H. Waldner, J. Beutel, M. Platzner, L. Thiele, and G. Tröster. The Case for Reconfigurable Hardware in Wearable Computing Nodes. In *ISWC 2002*, Piscataway, USA, October 2002.

[74] D. D. Polityko, M. Niedermayer, S. Guttowski, W. John, and H. Reichl. Drahtlose Sensornetze: Aspekte der Hardwareminiaturisierung. In *Fachgespräch Sensornetze*, Karlsruhe, Germany, March 2004.

[75] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Mobicom 2000*, Boston, USA, August 2000.

[76] C. Rathakumar, J. Mohamed, R. Raffaelli, and S. Walsh. A Downhole Noide Sensor Network. Technical report, University of California, San Diego, 2003.

[77] J. A. Rice. Undersea Networked Acoustic Communication and Navigation for Autonomous Mine-Countermeasure Systems. In *Intl. Symposium on Technology and the Mine Problem*, Monterey, USA, April 2002.

[78] R. Riem-Vis. Cold Chain Management using an Ultra Low Power Wireless Sensor Network. In *WAMES 2004*, Boston, USA, June 2004.

[79] H. Ritter, J. Schiller, and T. Voigt. Demand-based Location Determination in Wireless Sensor Networks. In *Adjuct Proc. EWSN 2004*, Berlin, Germany, January 2004.

[80] K. Römer. Time Synchronization in Ad Hoc Networks. In *MobiHoc 2001*, Long Beach, USA, October 2001.

[81] K. Römer. Temporal message ordering in wireless sensor networks. In *IFIP Mediterranean Workshop on Ad-Hoc Networks*, pages 131–142, Madhia, Tunisia, June 2003.

[82] K. Römer. The Lighthouse Location System for Smart Dust. In *MobiSys 2003*, San Franscisco, USA, May 2003.

[83] K. Römer. Tracking Real-World Phenomena with Smart Dust. In *EWSN 2004*, Berlin, Germany, January 2004.

[84] K. Römer, P. Blum, and L. Meier. Time Synchronization and Calibration in Wireless Sensor Networks. In I. Stojmenovic, editor, *Handbook of Sensor Networks : Algorithms and Architectures*. Wiley and Sons, September 2005.

[85] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, December 2004.

[86] K. Römer and F. Mattern. A unified view on space and time in sensor networks. *Elsevier Computer Communications*, 2005. To appear.

[87] J. S. Sandhu, A. M. Agogino, and A. K. Agogino. Wireless Sensor Networks for Commercial Lighting Control: Decision Making with Multi-Agent Systems. In *AAAI Workshop on Sensor Networks*, San Jose, USA, July 2004.

[88] C. Savarese, J. M. Rabaey, and K. Langendoen. Robust Positioning Algorithms for Distributed Ad-Hoc Wireless Sensor Networks. In *USENIX Annual Technical Conference*, Monterey, USA, June 2002.

[89] A. Savvides, C. C. Han, and M. Srivastava. Dynamic Fine-Grained Localization in Ad-Hoc Networks of Sensors. In *Mobicom 2001*, Rome, Italy, July 2001.

[90] A. Savvides, H. Park, and M. B. Srivastava. The n-hop multilateration primitive for node localization problems. *MONET*, 8:443–451, 2003.

[91] U. Schmid and K. Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12(2):173–228, 1997.

[92] L. Schwiebert, S. K. S. Gupta, and J. Weinmann. Research Challenges in Wireless Networks of Biomedical Sensors. In *Mobicom 2001*, Rome, Italy, July 2001.

[93] Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz. Localization from Mere Connectivity. In *ACM MobiHoc 2003*, Annapolis, USA, June 2003.

[94] Y. C. Shim and C. V. Ramamoorthy. Monitoring and Control of Distributed Systems. In *First Intl. Conference of Systems Integration*, pages 672–681, Morristown, USA, 1990.

[95] M. L. Sichitiu and C. Veerarittiphan. Simple, accurate time synchronization for wireless sensor networks. In *IEEE Wireless Communications and Networking Conference (WCNC'03)*, March 2003.

[96] F. Siegemund, C. Floerkemeier, and H. Vogt. The Value of Handhelds in Smart Environments. In *ARCS 2004*, Augsburg, Germany, March 2004.

[97] G. Simon, A. Ledeczi, and M. Maroti. Sensor Network-Based Countersniper System. In *Proc. SenSys*, Baltimore, USA, November 2004.

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

[99] M. Sung and A. S. Pentland. LiveNet: Health and Livestyle Networking Through Distributed Mobile Devices. In *WAMES 2004*, Boston, USA, June 2004.

[100] BEA Systems. MIUGS Proves Worth to U.S. Navy. *Information and Electronic Warefare Systems News*, 3(20), 2002.

[101] J. v. Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *2nd ACM International Workshop on Wireless Sensor Networks and Applications*, pages 11–19, September 2003.

[102] E. Vildjiounaite, E. J. Malm, J. Kaartinen, and P. Alahuhta. Location Estimation Indoors by Means of Small Computing Power Devices, Accelerometers, Magnetic Sensors, and Map Knowledge. In *PERVASIVE 2002*, Zurich, Switzerland, August 2002.

[103] R. Viswanathan and P. Varshney. Distributed Detection with Multiple Sensors: I. Fundamentals. *Proceedings of the IEEE*, 85(1):54–63, 1997.

[104] A. Ward, A. Jones, and A. Hopper. A New Location Technique for the Active Office. *IEEE Personal Communications*, 4(5):42–47, October 1997.

[105] B. A. Warneke, M. D. Scott, B. S. Leibowitz, L. Zhou, C. L. Bellew, J. A. Chediak, J. M. Kahn, B. E. Boser, and K. S. J. Pister. An Autonomous 16 cubic mm Solar-Powered Node for Distributed Wireless Sensor Networks. In *IEEE Sensors*, Orlando, USA, June 2002.

[106] M. D. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

[107] G. Wener-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring Volcanic Eruptions with a Wireless Sensor Networks. In *EWSN 2005*, Istanbul, Turkey, January 2005.

[108] K. Whitehouse and D. Culler. Calibration as Parameter Estimation in Sensor Networks. In *WSNA*, Atlanta, USA, September 2002.

[109] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A Wireless Sensor Network for Structural Monitoring. In *Sensys 2004*, Baltimore, USA, November 2004.

[110] Y. Xu, J. Heidemann, and D. Estrin. Geography-Informed Energy Conservation for Ad-Hoc Routing. In *MobiCom*, Rome, Italy, July 2001.

[111] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *IEEE Infocom 2002*, New York, USA, June 2002.

[112] ARGO - Global Ocean Sensor Network. www.argo.ucsd.edu.

[113] ARGUS - Advanced Remote Ground Unattended Sensor. www.globalsecurity.org/intell/systems/arguss.htm.

[114] Autarke Verteilte Mikrosysteme. www.pb.izm.fhg.de/hdi/020_projects/020_avm.

[115] Berkeley Motes. www.xbow.com/Products/Wireless_Sensor_Networks.htm.

[116] BTnodes. www.inf.ethz.ch/vs/res/proj/smart-its/btnode.html.

[117] CarTalk 2000. www.cartalk2000.net.

[118] Contaminant Transport. cens.ucla.edu/Research/Applications/ctm.htm.

[119] FleetNet. www.fleetnet.de.

[120] FTB Feintechnik Bertsch. www.ftb-bertsch.de.

[121] Intel Proactive Healthcare. intel.com/research/prohealth.

[122] Microclimate Monitoring. cens.ucla.edu/Research/Applications/habitat_sensing.htm.

[123] Monitoring Marine Microorganisms.
cens.ucla.edu/Research/Applications/momm.htm.

[124] National Semiconductors. www.national.com.

[125] Parapin - a Parallel Port Programming Library for Linux.
www.circlemud.org/~jelson/software/parapin/.

[126] Raumcomputer. www.raumcomputer.com.

[127] Seal Monitoring.
calvin.st-andrews.ac.uk/external_relations/news_article.cfm?reference=316.

[128] Seismic Monitoring.
cens.ucla.edu/Research/Applications/seismic_monitor.htm.

[129] SEWING - System for European Water Monitoring. www.sewing.mixdes.org.

[130] Shooter Localization.
www.isis.vanderbilt.edu/projects/nest/applications.html.

[131] Spec Mote. www.jlhlabs.com/jhill_cs/spec.

[132] Speckled Computing Consortium. www.specknet.org.

[133] Structural Health Monitoring of the Golden Gate Bridge.
www.cs.berkeley.edu/~binetude/ggb.

[134] The 29 Palms Experiment: Tracking vehicles with a UAV-delivered sensor network. tinyos.millennium.berkeley.edu/29Palms.htm.

[135] The Hogthrob project. www.hogthrob.dk.

[136] Vibration Monitoring.
intel.com/research/exploratory/wireless_sensors.htm#vibration.

[137] Whale Shark Monitoring. www.mcss.sc/whale.htm.

[138] WiseNet. www.csem.ch/wisenet.

Resume

Personal Data

June 16, 1972	Date of Birth
Freiberg	Birthplace
Germany	Citizenship

Education

1979–1981	*Juri-Gagarin-Schule*, Elementary School (Polytechnische Oberschule), Freiberg, GDR
1981–1989	*Wladimir-Iljitsch-Lenin-Schule*, Elementary School (Polytechnische Oberschule), class with extended language teaching, Freiberg, GDR
1989	*Geschwister-Scholl-Oberschule*, Senior Secondary School (Erweiterte Oberschule), Freiberg, GDR
Oct. 1989	Moved to western part of Germany
1990-1992	*Augustinergymnasium*, Senior Secondary School (Gymasiale Oberstufe), Friedberg, Germany
Apr. 1992	Abitur
1992–1999	Study of Computer Science (Informatik) at the *University of Frankfurt/Main*, Germany
Apr. 1995	Bachelor of Science (Vordiplom) in Computer Science
Aug. 1999	Master of Science (Diplom) in Computer Science
1999–2005	Ph.D. Student at the Department of Computer Science, *ETH Zurich*, Switzerland

Employment

1995–1999	Student Research Assistant at the Department of Computer Science, *University of Frankfurt/Main*, Germany
1999-2005	Research Assistant at the Department of Computer Science, *ETH Zurich*, Switzerland