Diss. ETH No. 16435

Interval-Based Time Synchronization for Mobile Ad-Hoc Networks

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH

for the degree of
Doctor of Sciences

presented by

LENNART LAURI RUDOLF MEIER
Dipl.-Inform., Saarland University, Germany
born 1976-03-15
citizen of
Finland and Germany

accepted on the recommendation of

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Friedemann Mattern, co-examiner

2006

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

TIK-SCHRIFTENREIHE NR. 78

Lennart Meier

Interval-Based Time Synchronization for Mobile Ad-Hoc Networks

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A dissertation submitted to the
Swiss Federal Institute of Technology Zurich
for the degree of Doctor of Sciences

Diss. ETH No. 16435

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Friedemann Mattern, co-examiner

Examination date: 2005-12-22

Abstract

In this thesis, we make a case for the use of guaranteed intervals for time synchronization in mobile ad-hoc networks. In particular, we look at wireless sensor networks (WSNs), a specific class of mobile ad-hoc networks. WSNs are envisioned to comprise a large number of small, inexpensive devices that operate on a very constrained energy budget.

Time synchronization is an important service in WSNs. Approaches developed in the distributed-systems field typically cannot be applied directly because of the limiting characteristics of WSNs: (a) There is no guarantee of stable connectivity between nodes. (b) Energy is a very scarce resource. Communication, which is needed to achieve and maintain synchronization, is expensive in terms of energy and hence has to be kept short. (c) Communication bandwidth is limited. (d) There is no a-priori configuration or infrastructure. In particular, there are few or even no reference clocks available.

In this thesis, we make a number of contributions to the state of the art in the field of time synchronization for mobile ad-hoc networks. Our main claim is that interval-based time synchronization is particularly suited for these networks. Specifically, our contributions are the following:

We present a new system model for the analysis of interval-based time synchronization in mobile ad-hoc networks. We justify why our abstractions are well chosen for this class of networks.

Using our system model, we derive worst-case bounds on the quality of interval-based synchronization and show the worst-case-optimality of a very simple algorithm.

The simple, worst-case-optimal algorithm is not optimal in the average case. We present three algorithms that are also worst-case-optimal but achieve better synchronization quality in the average case. We show that two of the algorithms achieve optimal synchronization, albeit at the cost of high memory and communication overhead. We describe how limiting the amount of data that is stored and communicated affects the synchronization quality.

We show that interval-based synchronization does not need particular communication patterns such as trees or clustered hierarchies. Hence, interval-based synchronization is resilient to mobility; our simulation results show that mobility actually improves it.

Finally, we derive a lower bound on the error of gradient clock synchronization in our system model.

Zusammenfassung

Der Kern dieser Arbeit ist die These, dass intervallbasierte Zeitsynchronisation besonders geeignet für mobile Ad-hoc-Netze ist. Im Besonderen betrachten wir drahtlose Sensornetze, eine spezifische Klasse mobiler Ad-hoc-Netze. Es wird allgemein angenommen, dass drahtlose Sensornetze aus einer Vielzahl kleiner, kostengünstiger Netzknoten mit sehr knappen Energieressourcen bestehen.

Zeitsynchronisation ist ein wichtiger Dienst in drahtlosen Sensornetzen. Ansätze aus dem Gebiet der verteilten Systeme können wegen der Einschränkungen drahtloser Sensornetze nicht direkt angewendet werden: (a) Es gibt keine Garantie für stabile Verbindungen zwischen Knoten. (b) Energie ist sehr knapp. Kommunikation, welche zur Erlangung und Aufrechterhaltung von Synchronisation nötig ist, ist energieintensiv und muss daher von kurzer Dauer sein. (c) Die Kommunikationsbandbreite ist beschränkt. (d) A-priori-Konfiguration der Netzknoten ist nicht möglich, und Infrastruktur ist nicht verfügbar. Insbesondere gibt es nur wenige oder gar keine Referenzuhren.

Diese Arbeit leistet einen Beitrag zum Stand der Forschung bezüglich Zeitsynchronisation in mobilen Ad-hoc-Netzen. Die zentrale Behauptung ist, dass intervallbasierte Synchronisation besonders geeignet für solche Netze ist. Die Beiträge dieser Arbeit im Detail:

Wir präsentieren ein neues Systemmodell für die Analyse intervallbasierter Zeitsynchronisation in mobilen Ad-hoc-Netzen. Wir begründen, weshalb unsere Abstraktionen für diese Klasse von Netzen gut gewählt sind.

Unter Verwendung unseres Modells leiten wir Schranken für die Qualität der intervallbasierten Synchronisation im ungünstigsten Fall her und zeigen, dass ein sehr einfacher Algorithmus optimal im ungünstigsten Fall ist.

Dieser einfache Algorithmus ist im durchschnittlichen Fall nicht optimal. Wir stellen drei Algorithmen vor, welche auch optimal im ungünstigsten Fall sind, und darüber hinaus bessere Synchronisation im durchschnittlichen Fall bieten. Wir zeigen die allgemeine Optimalität von zwei der drei Algorithmen, welche allerdings mit hohem Speicher- und Kommunikationsaufwand erkauft werden muss. Wir beschreiben die Auswirkungen einer Beschränkung der gespeicherten und kommunizierten Datenmenge auf die Synchronisationsqualität.

Wir zeigen, dass intervallbasierte Synchronisation nicht auf besondere Kommunikationsmuster wie Bäume oder gebündelte Hierarchien angewiesen ist. Daher ist sie Knotenmobilität gegenüber unempfindlich; unsere Simulationen zeigen, dass sie davon sogar profitiert.

Zuletzt leiten wir eine untere Schranke für den Fehler der sogenannten Gradientensynchronisation in unserem Systemmodell her.

Acknowledgments

This thesis describes joint work with Dr. Philipp Blum, Dr. Kay Römer, and Prof. Dr. Lothar Thiele. I want to thank them for the fruitful collaboration without which this thesis would not have been possible.

In particular, I would like to thank Prof. Dr. Lothar Thiele for supervising and guiding my research, and Prof. Dr. Friedemann Mattern for his willingness to be the co-examiner of my thesis.

The work presented in this thesis was supported in part by the National Centre of Competence in Research on Mobile Information and Communication Systems (NCCR MICS), which is funded by the Swiss National Science Foundation under grant number 5005-67322. This support is gratefully acknowledged.

Contents

1

Introduction

Time synchronization has been extensively studied in computer science, more specifically in the area of distributed systems. In this thesis, we make a case for the use of guaranteed intervals for time synchronization in mobile ad-hoc networks.

In this chapter, we first present time synchronization as an abstract problem, describe its peculiarities when implemented in a specific class of mobile ad-hoc networks, namely wireless sensor networks, and briefly characterize interval-based time synchronization. We then give an overview of the contributions and of the structure of this thesis.

1.1 Time synchronization for mobile ad-hoc networks

The time-synchronization problem

In everyday life, time synchronization is typically perceived as the following problem: A person—let us call her Alice—wants her watch to always show the legal time as precisely as possible. To achieve this, Alice obtains the legal time from an appropriate source, a so-called reference clock, and adjusts her watch to show the same time. The synchronization thus obtained does not last forever: Alice's imperfect watch does not measure time intervals with perfect accuracy, and hence its time will deviate more and more from the legal time as the time elapsed since the adjustment becomes larger. Alice therefore periodically synchronizes her watch with the reference clock. This illustrates that synchronization is an ongoing process; the higher the required accuracy of an imperfect

clock's time information, the more often the clock has to be synchronized to a reference clock.

In a more abstract sense, time synchronization means that various entities agree on the size of time intervals and possibly also on offsets on a common time scale; this makes time-coordinated actions possible. The entities use the means at their disposal, most notably imperfect clocks, to achieve synchronization. To emphasize the fact that it actually is the time information that is adjusted and communicated, we say "time synchronization" instead of "clock synchronization".

If all entities share the same resources for measuring time, synchronization is provided implicitly. For instance, two persons that have equal access to a common clock will clearly agree as to what the displayed time is. When entities do not share the time-measuring resources, time synchronization can only be achieved by communication. For example, assume that two persons, Alice and Bob, each only have access to a personal, local clock and to a communication channel between each other. Time synchronization can then be achieved for instance by Alice sending the reading of her clock to Bob, and Bob subsequently adjusting his clock to show the same time. Alternatively, Bob could leave his clock as it is and merely record the offset beween his and Alice's clock. As Alice's and Bob's clocks will typically exhibit different drifts, i.e. deviations from the ideal rate, periodical synchronization is necessary to keep the synchronization error (in this example: the difference between Alice's local time and what Bob believes is Alice's local time) within given limits.

A problem that does not emerge in everyday life but is very significant in distributed computer systems is that of the message-delay uncertainty. Suppose that the communication channel between Alice and a reference clock exhibits a delay that varies between 1 and 2 minutes. This means that when Alice receives a message with the time t_{send} when the reference clock sent the message, she can only deduce that the current legal time must be in the interval $t_{send} + [1\text{ min}, 2\text{ min}]$. Note that if Alice had a perfect (i.e. non-drifting) clock, she could measure the delay of the message. Likewise, if she knew the delay of the message, she could synchronize her clock perfectly. In the real world, clocks are imperfect and message delays are unknown.

Time synchronization in wireless sensor networks

Mobile ad-hoc networks are a fairly recent research area. While "mobile" refers to the fact that the network nodes can move, "ad hoc" means that these networks do not rely on existing infrastructure, but use whatever resources are available at the moment. Current cellular telephone networks that employ fixed base stations are therefore (partly) mobile, but not ad hoc. Portable computers with wireless networking capability according to the IEEE 802.11 standard can operate both in access-point mode (where they will open a connection only to infrastructure devices, namely to access points) or in ad-hoc mode (where they will connect to any other IEEE 802.11 device).

Wireless sensor networks (WSNs) are a specific class of mobile ad-hoc networks. They are envisioned to comprise a large number of small, inexpensive devices that operate on a very constrained energy budget. Sensors incorporated into these so-called sensor nodes collect data from their environment and transmit it through the ad-hoc network. Possible applications of WSNs include environmental monitoring, building technology, logistics, search-and-rescue operations, and battlefield use.

Time synchronization is an important service in sensor networks. For instance, the fusion of distributed sensor data typically requires information about the chronology of the sensor observations [Röm01]. In addition, the energy consumption of the sensor nodes can be reduced by synchronous power-on and shutdown of the radios of sender and receiver [CJBM01, IEE99, WESW98].

There has been a lot of research on time synchronization in the distributed-systems field [Mil]. Many of the existing results cannot be applied to sensor networks, since these networks have limiting characteristics [ER02]:

- *Robustness:* There is no guarantee of stable connectivity between nodes. This is due to the use of radio links, which are inherently less stable than cable connections, to sensor nodes being more likely to stop functioning because of depleted batteries or physical damage, and to the mobility of sensor nodes.

- *Energy efficiency:* Unlike traditional distributed systems, sensor nodes typically are powered by batteries or solar cells. This means that energy is a very scarce resource. Communication, which is needed to achieve and maintain synchronization, is expensive in terms of energy and hence has to be kept short.

- *Communication bandwidth:* Not only the duration, but also the bandwidth of communication is limited. This may at first seem no burden on time synchronization, as typical synchronization algorithms need to exchange only small amounts of data. But the limited bandwidth can result in temporary unavailability of the communication channel and hence in large message-delay uncertainties which hinder synchronization.

- *Ad-hoc deployment:* The clock-synchronization service must not rely on any a-priori configuration or infrastructure. In particular, there are few or even no reference clocks available.

For these reasons, well-known algorithms such as NTP [Mil91] are not applicable in wireless sensor networks. Various algorithms tailored specifically to these networks have been proposed recently; we give a comprehensive overview in Chapter 2.

1.2 Contributions

In this thesis, we make a number of contributions to the state of the art in the field of time synchronization for mobile ad-hoc networks. Our main claim is that interval-based time synchronization is particularly suited for these networks. We will now briefly characterize interval-based synchronization and then give a detailed overview of our contributions.

Interval-based time synchronization

We are used to time information being given as a single value indicating a point in time. The time that Alice reads on her watch is an estimate that has a certain error with respect to the legal time. Interval-based synchronization provides guaranteed bounds on points in time. It was first proposed—in the context of traditional distributed systems—in [MO83] and was extensively studied in [SS97]. Time information is represented using an interval $[T^l, T^u]$ defined by guaranteed lower and upper bounds T^l and T^u on the real time[1]. We call the size $\Delta T = T^u - T^l$ of the interval the *uncertainty*. The goal of an interval-based synchronization algorithm is to maintain guaranteed lower and upper bounds T^l and T^u while keeping the uncertainty as small as possible.

If Alice's watch was interval-based, it could show a lower bound T^l and an upper bound T^u such that the current legal time t is always guaranteed to lie within the two: $T^l \leq t \leq T^u$. It may seem that the elimination of the error with respect to the legal time comes at the cost of the introduction of uncertainty, and that not much truly changes. However, we will show in this thesis that time intervals provide various advantages over single-value estimates.

Detailed contributions

New system model
We present a new system model for the analysis of interval-based time synchronization in mobile ad-hoc networks. We justify why our abstractions are well chosen for this class of networks.

Worst-case bounds
Using our system model, we derive worst-case bounds on the quality of interval-based synchronization, namely lower bounds on the synchronization error for a given communication pattern. We show that a very simple algorithm from [MO83] is worst-case-optimal.

Optimal and efficient interval-based synchronization
The simple, worst-case-optimal algorithm from [MO83] is not optimal in the average case. We present three algorithms that are also worst-case-optimal but achieve better synchronization quality in the average case. We show that two of the algorithms achieve optimal synchronization, albeit at the cost of high

[1] In this thesis, the terms legal time, real time, and physical time are interchangeable.

memory and communication overhead. We describe how limiting the amount of data that is stored and communicated affects the synchronization quality.

Mobility
We show that interval-based synchronization does not need particular communication patterns such as trees or clustered hierarchies. This makes the interval-based approach resilient to node mobility, as there are no broken topologies that have to be repaired. Our simulation results suggest that mobility actually improves interval-based synchronization by increasing the rate of information dissemination through the network.

Gradient clock synchronization in wireless sensor networks
Gradient clock synchronization was introduced in [FL04] and can briefly be characterized as synchronization that maintains the gradient property. This property requires the difference between any two network nodes' clocks to be upper-bounded by a non-decreasing function of their distance. This means that every node has to be synchronized better to nearby nodes than to faraway nodes. The system model in [FL04] differs considerably from our system model. We show that a similar lower bound for the synchronization error as in [FL04] exists also in our system model.

1.3 Overview

This thesis is structured as follows: In Chapter 2, we give an overview of the principles and the state of the art of synchronization in wireless sensor networks. These principles apply also to mobile ad-hoc networks in general. In Chapter 3, we present our results regarding interval-based synchronization. Gradient clock synchronization in wireless sensor networks is addressed in Chapter 4.

2

Time Synchronization in Wireless Sensor Networks

2.1 Introduction

In this chapter, we give an overview of the principles and of the state of the art of time synchronization in wireless sensor networks. Our aim is both to give a comprehensive picture of the various existing approaches and to illustrate that there are many degrees of freedom in the design, implementation, and evaluation of synchronization algorithms.

In Section 2.2, we discuss applications of synchronized time in sensor networks, present the particular challenges that sensor networks pose, and discuss why traditional synchronization approaches fail to meet these challenges. Section 2.3 presents models of sensor nodes, of hardware clocks, and of communication. Section 2.4 gives an overview of the various classes of synchronization. In Section 2.5, we present common synchronization techniques. Section 2.6 examines current synchronization algorithms. Section 2.7 presents common techniques for evaluating synchronization algorithms and selected evaluation results.

This chapter is based on [RBM05].

2.2 Applications of Time and Challenges in WSNs

Sensor networks are used to monitor real-world phenomena. For such monitoring applications, physical time often plays a crucial role. We will discuss these

applications of time in Section 2.2.1. Providing synchronized physical time is a complex task due to various challenging characteristics of sensor networks. In Section 2.2.2, we present these challenges and discuss why synchronization algorithms for traditional distributed systems often do not meet them.

2.2.1 The need for synchronized time

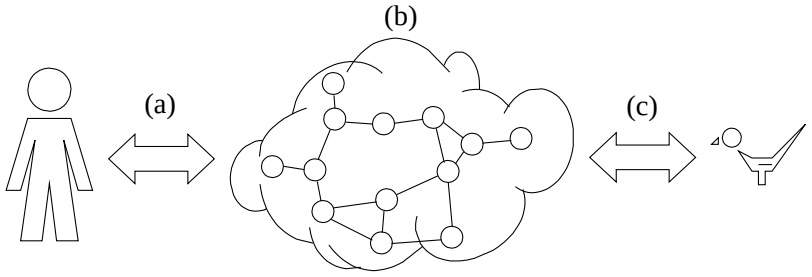

Fig. 1: Applications of physical time: (a) interaction of an external observer with the sensor network, (b) interaction among sensor nodes, and (c) interaction of the sensor network with the real world.

Physical time plays a crucial role for many sensor-network applications. While many traditional applications of time also apply to sensor networks, we will focus here on areas specific to sensor networks. Figure 1 illustrates a rough classification of applications of physical time: (a) at the interface between the sensor network and an external observer, (b) among the nodes of the sensor network, and (c) at the interface between the sensor network and the observed physical world. In the following paragraphs, we will discuss applications of time in these three domains. Note that some applications are hard to assign to a single domain. In such cases, we picked the most appropriate domain.

Sensor network – observer

In many applications, a sensor network interfaces to an external observer for tasking, reporting results, and management. This observer may be a human operator or an autonomous computing system. Tasking a sensor network often involves the specification of time windows of interest such as “only during the night”. As a sensor network reports observation results to an external observer, temporal properties of observed physical phenomena may be important. For example, the times of occurrence of physical events are often crucial for the observer to associate event reports with the originating physical events. Physical time is also necessary for determining properties such as speed or acceleration.

Sensor network – real world

In sensor networks, many sensor nodes may observe a single physical phenomenon. One of the key functions of a sensor network is hence the assembly of a number of distributed observations into a single, coherent estimate of the original phenomenon; this process is known as data fusion. For example, if sensors can only detect the proximity of an object, then higher-level information (such

as speed, size, or shape) can be obtained by correlating data from multiple sensor nodes. Time is a crucial ingredient for data fusion: For instance, the speed of a mobile object can be estimated by the quotient of the spatial and temporal distances between two consecutive sightings of the object by different nodes.

Since many instances of a physical phenomenon can occur within a short time, one of the tasks of a sensor network is the separation of sensor samples, i.e. the partitioning of sensor samples into groups that each represent a single physical phenomenon. Temporal relationships (e.g., distance in time) between sensor samples are a key element for separation.

Temporal coordination among sensor nodes may also be necessary to ensure correctness and consistency of distributed measurements [GRWE04]. For example, if the sampling rate of sensors is low compared to the frequency of an observed phenomenon, it may be necessary to ensure that sensor readout occurs concurrently at all sensor nodes to avoid false observation results.

It is anticipated that large-scale, complex actuation functions will be implemented by coordinated use of many simple actuator nodes. This requires temporal coordination.

Within the sensor network

Time is also a valuable tool for intra-network coordination among different sensor nodes. Many applications of time known from traditional distributed systems also apply to wireless sensor networks. [Lis91] points out a number of applications of time in distributed systems such as 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 of concurrency control is the use of time-division multiplexing in wireless communication, where multiple access to the shared communication medium is achieved by assigning time slots to the communicating nodes. This requires the participating sensor nodes to share a common view of time.

A number of approaches try to improve energy efficiency by frequently switching nodes or components thereof into power-saving sleep modes (e.g., [YHE02]). To nonetheless ensure seamless operation of the sensor network, temporal coordination of the sleep periods among sensor nodes is required.

Another important service for sensor-network applications is temporal message ordering (e.g., [Röm03]). Many data-fusion algorithms have to process sensor readings ordered by the time of occurrence (e.g., the approach for speed estimation sketched above). However, the highly variable message delays in sensor networks imply that messages from distributed sensor nodes may often not arrive at a receiver in the order in which they were sent. Reordering messages according to the time of sensor readout requires temporal coordination among sensor nodes.

Methods for localization of sensor nodes based on the measurement of time of flight or difference of arrival time of certain signals also require synchronized time (e.g., [GBEE02]).

2.2.2 Revisiting time synchronization for sensor networks

Time synchronization is a research area with a very long history. Over time, numerous algorithms have been proposed and have been in large-scale use. The Network Time Protocol (NTP) [Mil91] is perhaps one of the most advanced and time-tested systems. However, several characteristics of sensor networks often preclude the use of existing synchronization techniques in this domain.

In the following, we discuss sensor-network challenges that impact the design of time-synchronization approaches. Using NTP as an example, we will outline why traditional approaches often do not meet the requirements of sensor networks (see also [ER02]). Note that some of the illustrated shortcomings of NTP are relatively easy to fix, while others are not. To provide the necessary background, we will first give an overview of NTP.

NTP was designed for large-scale networks with a rather static topology (such as the Internet). Nodes are 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 the Global Positioning System (GPS) (which provides global time with a precision significantly below 1 µs). 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 during the configuration of the node. Nodes frequently exchange synchronization messages with their parents and use the obtained information to adjust their clocks by regularly incrementing them.

Energy and other resources

Sensor-network applications often require sensor nodes to be small and cheap. This has a number of important implications. First of all, the amount of energy that can be stored in or scavenged by small devices is typically very limited due to the low energy density of available and foreseeable technology. To ensure longevity despite a limited energy budget, both hardware and software have to be designed with energy efficiency as a dominating goal. Also the computing, storage, and communication capabilities of a single sensor node are rather limited due to size and energy constraints.

These 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 infrastructure-based 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, nodes employing NTP are continuously synchronized with maximum precision, although 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 energy overhead in sensor networks. For example, NTP maintains a synchronized system clock by regularly adding small increments to the system-clock counter. This 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

Due to their deployment in the physical environment, sensor networks are subject to a high degree of network dynamics. Sensor nodes can be mobile, die due to depleted batteries or due to environmental influences, and new sensor nodes may be added at any point in time. This results in relatively frequent and unpredictable changes in the network topology and possibly even in (temporary) network partitions. Mobile nodes can transport messages across partition boundaries by storing a received message and forwarding it as soon as a new partition is entered. The end-to-end delay of such message paths is very unstable and hard to predict.

The operation of NTP is 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. Due to the above-mentioned effects, 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 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. Consider for example an application where mobile sensor nodes with sporadic network connectivity time-stamp sensor readings and deliver these records to an observer as they pass by a base station (e.g., [JOW+02]). The base station may then want to compare time stamps generated by different sensor nodes in order to evaluate the collected data. However, in the above scenario, there might not be a network connection between the various originators of the time-stamped messages at any point in time. Hence, NTP cannot be applied in such settings.

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. For example, under dense foliage or inside buildings, GPS cannot be used since there is no free line of sight to the GPS satellites.

In order to improve the precision and availability of synchronization in large networks, NTP injects the reference time at many points into the network. Hence, any node in the network is likely to find a source of reference time within a distance of only a few hops. Note that shorter paths tend to be more reliable and more predictable, since they include fewer sources of error and unpredictability.

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 large, leading to reduced precision. This is particularly problematic when collocated sensor nodes require very precise mutual synchronization, for example to cooperate in observing a nearby physical event. With a single master node, the collocated nodes may end up using different synchronization paths, which results in different synchronization errors (i.e., time offsets) with respect to the master node and hence in poor mutual synchronization.

Configuration
After initial deployment, it is often infeasible to physically access the sensor nodes for hardware or software maintenance. The large number of nodes also precludes manual configuration of individual nodes. While traditional networks such as the Internet do also consist of a large number of nodes, there is an accordingly large number of network administrators, each taking care of a manageable number of computers. With sensor networks, however, half a dozen of human operators may be responsible for thousands of sensor nodes.

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.

2.3 System Model

In the sections ahead, we will analyze various synchronization approaches. We will now specify the system model that we use as the foundation of our analysis. First, we describe how we model clocks. We then specify the characteristics of communication between nodes in a sensor network.

All our modeling is done in terms of discrete time and events. An event can represent communication between nodes, a sensor measurement, the injection of time information at a node, etc. We denote the real time at which event a occurs as t_a, and the local time of node N_i at that time as h_a^i. Note that our model does not explicitly contain node mobility or network dynamics; these aspects are included implicitly by the absence or existence of corresponding communication events.

2.3.1 Clock models

Digital clocks measure time intervals. They typically consist of a counter h (which we will also refer to as "the (local) clock") that counts time steps of an ideally fixed length; we denote the reading of the counter at real time t as $h(t)$. The counter is incremented by an oscillator with a rate (or frequency) f.

The rate f at time t is given as the first derivative of $h(t)$: $f(t) = \mathrm{d}h(t)/\mathrm{d}t$. An ideal clock would have rate 1 at all times, but the rate of a real clock fluctuates over time due to changes in supply voltage, temperature, etc. If the fluctuation were allowed to be arbitrary, the clock's reading would obviously give no information at all. Fortunately, it is limited by known bounds. Different types of bounds on the rate fluctuation lead to different clock models:

Constant-rate model
The rate is assumed to be constant. This is reasonable if the required precision is small compared to the rate fluctuation, i.e. if the error introduced by ignoring the rate fluctuation is negligible.

Bounded-drift model
The deviation of the rate from the ideal rate 1 is assumed to be bounded. We call this deviation the clock's *drift* $\rho(t) = f(t) - 1 = \mathrm{d}h(t)/\mathrm{d}t - 1$, and denote the corresponding bound with $\hat{\rho}$:

$$-\hat{\rho} \leq \rho(t) \leq \hat{\rho} \quad \forall t \ . \tag{2.1}$$

A reasonable additional assumption is $\rho(t) > -1$ for all times t. This means that a clock can never stop ($\rho(t) = -1$) or run backward ($\rho(t) < -1$).

If two events a, b with $t_a < t_b$ occur at a node N_i whose clock's drift ρ_i is bounded according to (2.1), then node N_i can compute lower and upper bounds $\Delta_i^l[a,b]$, $\Delta_i^u[a,b]$ on the real-time difference $\Delta[a,b] := t_b - t_a$ as:

$$\Delta_i^l[a,b] := \frac{h^i(t_b) - h^i(t_a)}{1+\hat{\rho}} \qquad \Delta_i^u[a,b] := \frac{h^i(t_b) - h^i(t_a)}{1-\hat{\rho}} \ . \tag{2.2}$$

This model is typically reasonable, since bounds on the oscillator's rate are given by the hardware manufacturer. Sensor nodes usually contain non-expensive oscillators, and thus we have $\hat{\rho} \in [10\text{ ppm}, 100\text{ ppm}]$[1]. Note that in this model, the drift can jump arbitrarily within the bounds specified in (2.1). The next model limits the variation of the drift.

Bounded-drift-variation model
The variation $\vartheta(t) = \mathrm{d}\rho(t)/\mathrm{d}t$ of the clock drift is assumed to be bounded:

$$-\hat{\vartheta} \leq \vartheta(t) \leq \hat{\vartheta} \quad \forall t \ . \tag{2.3}$$

This assumption is reasonable if the drift is influenced only by gradually changing conditions such as temperature or battery voltage. Using (2.3), a node can estimate its current drift and compute bounds on its drift for future times. We refer to this as *drift compensation*. Note that drift compensation is sometimes defined as the adjustment of bounds according to (2.2), e.g. in [SS97].

Of course, combined models are also possible: we can assume both (2.1) and (2.3).

[1] The abbreviation "ppm" stands for "parts per million", i.e. 10^{-6}. A clock with a drift of 100 ppm drifts 100 s in 1000000 s, or 100 µs in 1 s.

2.3.2 Software clocks

A synchronization algorithm can either directly modify the local clock h, or alternatively construct a software clock c. A software clock is a function taking a local clock value $h(t)$ as input and transforming it to the time $c(h(t))$. This time is the final result of synchronization, and we therefore call it the synchronized time. For example, $c(h(t)) = t_0 + h(t) - h(t_0)$ is a software clock that starts with the correct real time t_0 and then runs with the same speed as the local clock h. In general, we require that a software clock is a piecewise continuous[2], strictly monotonically increasing function.

2.3.3 Communication models

Communication is needed to achieve and maintain synchronization. We now identify different communication parameters that affect synchronization.

Unicast vs. multicast

If a message is sent by one network node and is received by at most one other network node, we call this unicast or point-to-point communication. Multicast communication occurs when a message is sent by one network node and is received by an arbitrary number of other network nodes. The case where all nodes within transmission range are recipients is called broadcast. Wireless sensor networks typically use simple broadcast radios, such that a sensor node's transmission is overheard by all nodes within its transmission range.

Symmetrical vs. asymmetrical links

If node A can receive messages sent by node B if and only if node B can receive messages sent by node A, we say that the link between these two nodes is symmetrical. Otherwise, it is asymmetrical. An example for an asymmetrical link is the link between a base station with high transmit power and a mobile device with low transmit power: Beyond a certain distance between the two, only communication in direction from the base station to the mobile device is possible. Wireless sensor networks are typically envisioned to consist of a large number of small sensor nodes, and a small number of more powerful (regarding energy, memory, processing power, and transmit power) nodes. The links between these two types of nodes would be asymmetrical.

Implicit vs. explicit synchronization

When comparing clock-synchronization approaches, it is important to distinguish whether synchronization information can be sent only with the messages that the sensor-network application transmits ("piggyback"), or whether additional communication (i.e., messages sent only for the sake of synchronization) is allowed. There is a trade-off between the amount of additional communication and the achievable synchronization quality. Additional communication incurs additional energy consumption and can reduce the bandwidth available for application data. Piggybacked time information does typically not reduce bandwidth significantly, since there are no additional message headers to be

[2] A function is piecewise continuous if it is continuous on all but a finite number of points.

transmitted or transmission slots to be occupied, and since the time information is small in size.

Delay uncertainty
As far as synchronization is concerned, the goal of communication is to convey time information. The delay of the messages sent between nodes has to be taken into account when extracting this time information; we will explore this in Section 2.5.1. The message delay consists of

- the send time, lasting from when the application issues the send command to when the node starts trying to send; it is caused by kernel processing, context switches, and system calls, and hence varies with the system load,
- the (medium) access time, lasting from when the node is ready to send to when it starts the transmission; this is the time that is spent waiting for access to the wireless channel, and hence varies with the network load,
- the propagation time, which is the time it takes for the radio signal to travel from the sender to the receiver; it is constant for any pair of nodes with constant distance, and is negligible compared to the other delay components in wireless sensor networks (since distances are small and radio signals travel very fast), and
- the receive time, lasting from the reception of the signal to the arrival of the data at the application; the same factors causing the send time apply.

The send and receive time (and especially the uncertainty about them) can be reduced by implementing the time-stamping of outgoing and incoming messages at a very low level, for instance in the MAC layer. Radios employing synchronous techniques such as frequency hopping or time-division multiplexing could in principle provide implicit synchronization with small delay uncertainties at the physical layer, but commonly access to this layer is not assumed in the design of synchronization algorithms.

Overall, message-delay uncertainties in typical wireless sensor networks are rather large compared to those in wired networks. This is due to the lower link reliability and bandwidth (see Section 2.2.2).

2.4 Classes of Synchronization

Synchronization is commonly understood as “making clocks show the same time”, but there are actually many different types of synchronization. In the following, we will give an overview of the various choices available for synchronization. When choosing the synchronization approach for a given sensor-network application, the maxim is to fulfill the application’s requirements with the smallest possible effort in terms of computation, memory, and especially energy.

2.4.1 Internal vs. external

The synchronization of all clocks in the network to a time supplied from outside the network is usually referred to as *external* synchronization. NTP performs external synchronization, as do sensor nodes synchronizing their clocks to a master node. Note that it makes no difference whether the source of the common system time is also a node in the network or not. External synchronization to the physical time is needed for the correct measurement of physical quantities.

Internal synchronization is the synchronization of all clocks in the network, without a predetermined master time. The only goal here is consistency among the network nodes. External synchronization requires consistency within the network *and* with respect to the externally provided system time. Internal synchronization is necessary and sufficient for time-division multiplexing in wireless communication, where multiple access to the shared communication medium is achieved by assigning time slots to the communicating nodes.

In everyday life, we are mostly faced with external synchronization, namely with keeping watches and clocks in computers, cell phones, PDAs, cars, microwave ovens, etc. synchronized to the legal time. Groups of persons that want to collaborate in a synchronized manner only need internal synchronization of their clocks.

2.4.2 Continuous vs. on-demand

The *lifetime* of synchronization is the period of time during which synchronization is required to hold. If time synchronization is *continuous*, the network nodes strive to maintain synchronization (of a given quality) at all times. For some sensor-network applications, *on-demand* synchronization can be as good as continuous synchronization in terms of achieving the required synchronization quality, but much more efficient. During the (possibly long) periods of time between events of interest, no synchronization is needed, and communication and hence energy consumption can be kept at a minimum. As the time intervals between successive events become shorter, a break-even point is reached where continuous and on-demand synchronization perform equally well. There are two kinds of on-demand synchronization:

Event-triggered on-demand synchronization is based on the idea that in order to time-stamp a sensor event, a sensor node needs a synchronized clock only immediately after the event has occurred. It can then compute the time-stamp for the moment in the recent past when the event occurred. Post-facto synchronization [EGE02] is an example of event-triggered synchronization.

Time-triggered on-demand synchronization is what we use if we are interested in obtaining sensor data from multiple sensor nodes for a specific time. This means that there is no event that triggers the sensor nodes, but the nodes have to take a sample at precisely the right time. This can be achieved via *immediate* synchronization (where sensor nodes receive the order to immediately take a sample and time-stamp it) or *anticipated* synchronization (where the order is to take the sample at some future time, the *target time*). Anticipated synchro-

nization is necessary if rapid and simultaneous transmission of the order to all involved sensor nodes cannot be guaranteed. This is especially the case if sensor nodes are more than one hop away from the node giving the order.

Note that for successful anticipated synchronization, it suffices to maintain a synchronization quality which guarantees that the target time is not missed. This means that the required synchronization quality grows as the real time approaches the target time. There is no need to synchronize with maximum quality right from the beginning.

Analogously to the event-triggered post-facto synchronization, we might refer to time-triggered synchronization as pre-facto synchronization.

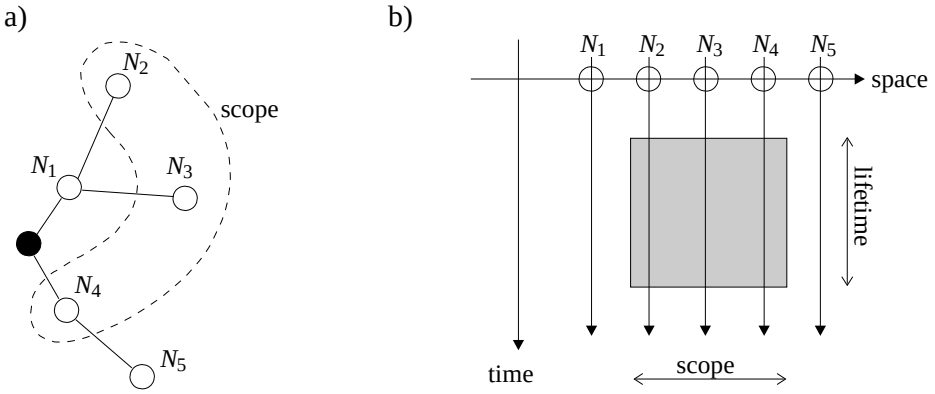

Fig. 2: Scope and lifetime define where and when synchronization is required. (a) shows the topology of some network, (b) illustrates scope and lifetime of the synchronization: Only nodes N_2, N_3 and N_4 need synchronization.

2.4.3 All nodes vs. subsets

The *scope* of synchronization defines which nodes in the network are required to be synchronized. Depending on the application, the scope comprises all or only a subset of the nodes. Event-triggered synchronization can be limited to the collocated subset of nodes which observe the event in question.

2.4.4 Rate synchronization vs. offset synchronization

Rate synchronization means that nodes measure identical time-interval lengths. In a scenario where sensor nodes measure the time between the appearance and disappearance of an object, rate synchronization is sufficient and necessary for comparing the duration of the object's presence within the sensor range of different nodes, but not for ordering the observations chronologically.

Offset synchronization means that nodes measure identical points in time, i.e. at some time t, the software clocks of all nodes in the scope show t. Offset synchronization is needed for combining time stamps from different nodes.

The difference between rate and offset synchronization is illustrated in Figure 3. Node N_2 can compute the bird's speed all by itself by dividing the distance

between the bird's positions at events *a* and *b* by the corresponding local-time difference. For this, the node's clock must be rate-synchronized to the real-time rate 1. Alternatively, data from nodes N_2 and N_3 can be combined to compute the bird's speed, e.g. by using events *b* and *c*. The nodes' clocks have to be offset-synchronized for this.

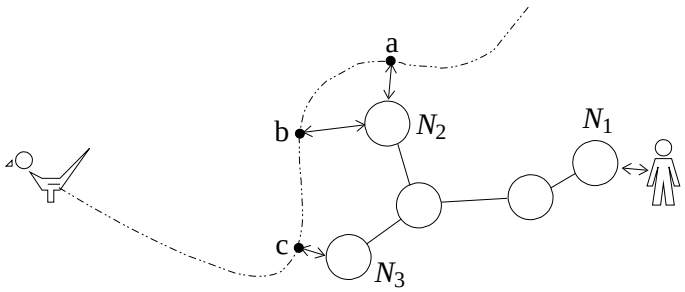

Fig. 3: At events *a*, *b*, and *c*, nodes N_2 and N_3 measure the position of the bird and time-stamp this data with their current local time. Rate or offset synchronization is needed depending on how the data from the three events is to be combined.

2.4.5 Timescale transformation vs. clock synchronization

Time synchronization can be achieved in two fundamentally different ways. We can synchronize clocks, i.e. make all clocks display the same time at any given moment. To achieve this, we have to perform rate and offset synchronization (or continuous offset synchronization, which however is costly in terms of energy and bandwidth and requires reliable communication links). The other approach is to transform timescales, i.e. to transform local times of one node into local times of another node.

Both approaches are equal in the sense that if we have either perfect clock synchronization or perfect timescale transformation, the distributed sensor data can be combined as if it had been collected by a single node. The approaches differ in that clock synchronization requires either communication across the whole network (for internal synchronization) or some degree of global coordination (for external synchronization). This calls for communication over multiple hops (which however tends to degrade synchronization quality), or well-distributed infrastructure which for instance guarantees that every sensor node is only a few hops away from a node equipped with a GPS receiver. Timescale transformation does not have these drawbacks, but may instead incur additional computation and memory overhead.

We illustrate the difference between clock synchronization and timescale transformation using the example shown in Figure 3. If the clocks of nodes $N_1, \ldots, N_3$ are synchronized, node N_1 can directly combine the sensor data from nodes N_2 and N_3, since the time stamps refer to the same timescale. If the clocks are not synchronized, a timescale transformation on the received time stamps is necessary. The final result is identical to that of using synchronized clocks.

2.4.6 Time instants vs. time intervals

Time information can be given by specifying time instants (e.g., "$t = 5$") or time intervals ("$t \in [4.5, 5.5]$"). In both cases, the time information can be refined by adding a statement about its quality. For instance, the time information may be guaranteed to be correct with a certain probability, or even probability distributions for the time can be given. A measure for the quality of the time information can then be defined; we will speak of its inverse, the *time uncertainty*.

In sensor networks, using guaranteed time intervals can be very attractive. Interestingly, this approach has not received much attention, although it has a number of advantages over using time instants, as we will see in Chapter 3.

2.5 Synchronization Techniques

In this section, building blocks and fundamental mechanisms of time synchronization algorithms are presented. The section is organized by increasing complexity: In Section 2.5.1, various approaches for obtaining a single reading of the clock of a remote node are presented. In Section 2.5.2, techniques for maintaining synchronization are discussed. In Sections 2.5.3 and 2.5.4, we show how multiple samples can improve synchronization between two nodes. Finally, various approaches to organize the synchronization process in larger networks are discussed in Section 2.5.5.

2.5.1 Taking one sample

Assume the simple scenario shown in Figure 4 (a), with two nodes N_i and N_j that can exchange messages. Synchronization of these nodes means that the nodes establish some relationship between their local clocks h^i and h^j.

Unidirectional synchronization

The conceptionally simplest solution is illustrated in Figure 4 (b). Node N_i sends a message containing a local time stamp h^i_a to node N_j, where it is received at local time h^j_b. Node N_j cannot determine the delay d of the message. It only knows that the local clock of node N_i showed h^i_a before its own local clock shows h^j_b. Thus, its local time when the message was sent was $h^j_a < h^j_b$, and node N_i's local time when the message is received is $h^i_b > h^i_a$. Time synchronization consists of estimating either h^i_b or h^j_a.

If a-priori bounds on the message delay d are known, i.e. if $d_{\min} \leq d \leq d_{\max}$, then the estimations

$$h^j_a \approx h^j_b - 1/2(d_{\min} + d_{\max}) \qquad h^i_b \approx h^i_a + 1/2(d_{\min} + d_{\max})$$

minimize the synchronization error in the worst case. Alternatively, $h^j_b - d_{\max}$ and $h^j_b - d_{\min}$ are lower and upper bounds on h^j_a (and $h^i_a + d_{\min}$ and $h^i_a + d_{\max}$ are bounds on h^i_b).

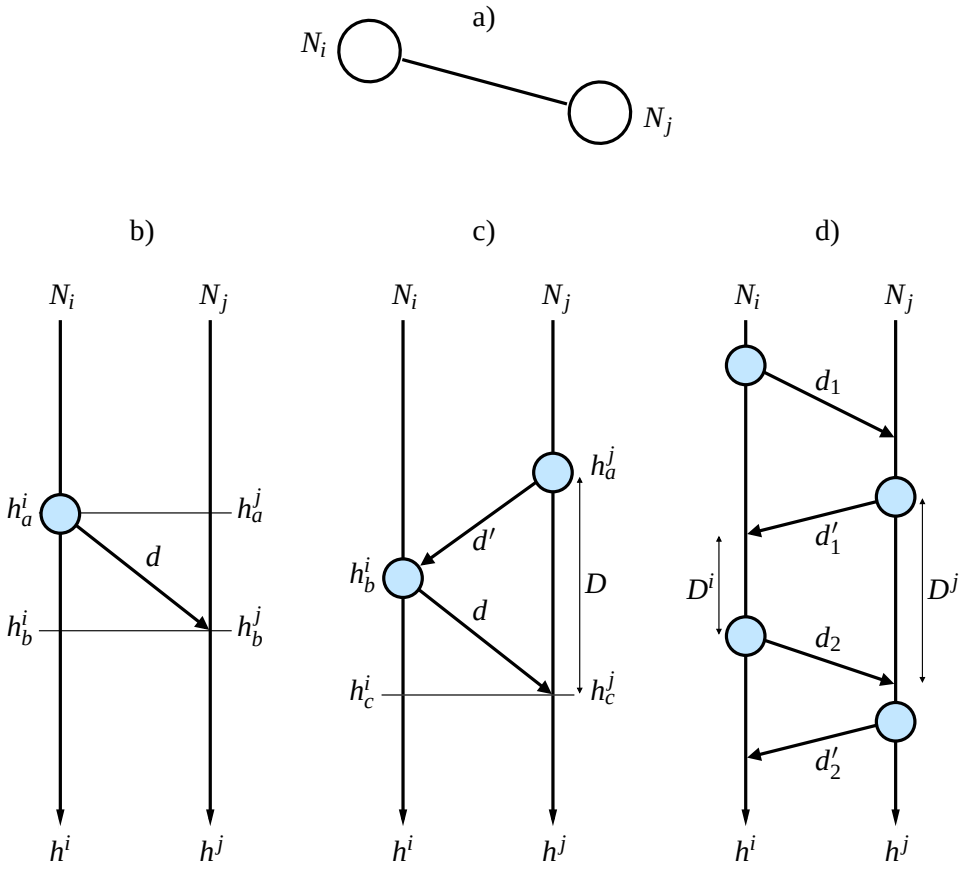

Fig. 4: *Uni- and bidirectional synchronization.* a) Node N_j determines the offset of its local clock relative to that of node N_i using uni- (b) or bidirectional (c,d) communication. In contrast to scheme c, scheme d allows both nodes to measure a round-trip time.

Round-trip synchronization

A slightly more complex solution is illustrated in Figure 4 (c): Node N_j sends a query message to node N_i, asking for the time stamp h_b^i. Node N_j measures the round-trip time $D = h_c^j - h_a^j$, i.e. the length of the time interval between sending the request and receiving the reply. Without a-priori knowledge, node N_j now knows that the delay d of the received time stamp is bounded by 0 and D. If a-priori bounds $d_{\min}$, $d_{\max}$ with $d_{\min} \leq d \leq d_{\max}$ on the message delay d are known, then node N_j knows that d is bounded by $\max(D - d_{\max}, d_{\min})$ and $\min(d_{\max}, D - d_{\min})$.

The estimation $h_b^j \approx h_c^j - D/2$ minimizes the worst-case synchronization error; $h_c^j - (D - d_{\min})$ and $h_c^j - d_{\min}$ are lower and upper bounds on h_b^j. Similarly, an estimation and bounds for h_c^i can be determined.

In comparison to the unidirectional approach, round-trip synchronization has the advantage of providing an upper bound on the synchronization error. The mechanism known as *probabilistic* time synchronization [Cri89] uses this to decrease the synchronization error as follows: After receiving the reply mes-

sage, N_j checks whether the worst-case synchronization error $D/2 - d_{\min}$ is below a specified threshold. If it is not, N_j sends a new request message to N_i. This procedure is repeated until the reception of a pair of request and reply messages that achieves the required synchronization error. The smaller the threshold, the more messages have to be exchanged on average.

The main disadvantage of round-trip synchronization is that the amount of messages increases linearly with the number of nodes that communicate with N_i, while in the unidirectional case, a single broadcast message sent by N_i can serve an arbitrary number of nodes. A combination of the advantages of both approaches is known as *eavesdropping* or *anonymous synchronization* and was first described in [DRSW95]. The basic idea is the following: Node N_j sends a broadcast message to N_i and some additional node N_k; N_i replies with a broadcast message to N_j and N_k. Node N_k assumes that the second message was produced after it had received the first message, thus node N_k can do round-trip synchronization with the two local receive time stamps and the send time stamp from N_i without ever producing any messages itself.

In Figure 4 (d), two modifications of round-trip synchronization are illustrated. Firstly, it is not necessary that N_i replies immediately to query messages. Node N_i can instead measure the duration D^i between receiving the query message and sending the reply, and node N_j can then account for this duration in its calculations. Secondly, the message exchange shown in Figure 4 (c) is asymmetrical, i.e. only N_j can do round-trip synchronization. Therefore, at least one additional message from N_j to N_i is required for N_i to be able to estimate or bound remote time stamps.

Reference broadcasting

A third approach is shown in Figure 5. In addition to nodes N_i and N_j, a so-called *beacon* node N_k is involved. The beacon sends a broadcast message to the other nodes. The delays d (to N_i) and d' (to N_j) are almost equal. N_i then sends the time stamp h_a^i to N_j. Node N_j measures the length of the time interval $D = h_b^j - h_{a'}^j$ between the arrivals of the two messages and can then estimate $h_b^i \approx h_a^i + D$.

This approach was first proposed in [HS91] under the name *a-posteriori agreement*. It became more widely known in the sensor-network community as *reference-broadcast* synchronization (RBS) [EGE02]. Its main advantage is that a broadcast message is received almost concurrently (even though its delay is largely variable), and thus the synchronization error typically is smaller than with unidirectional or round-trip synchronization.

The reference-broadcast technique can be used in many variations. For example, Figure 5 (c) shows a solution presented in [DH04] for the case that nodes N_i and N_j, while being able to receive messages from N_k, cannot communicate directly with each other. N_j replies to N_k, which then can estimate its own local time $h_{a'}^k$ and send this information in another broadcast message to N_i and N_j. In [EGE02], yet another version is described: All nodes report their time stamps to a single node, which then broadcasts all information.

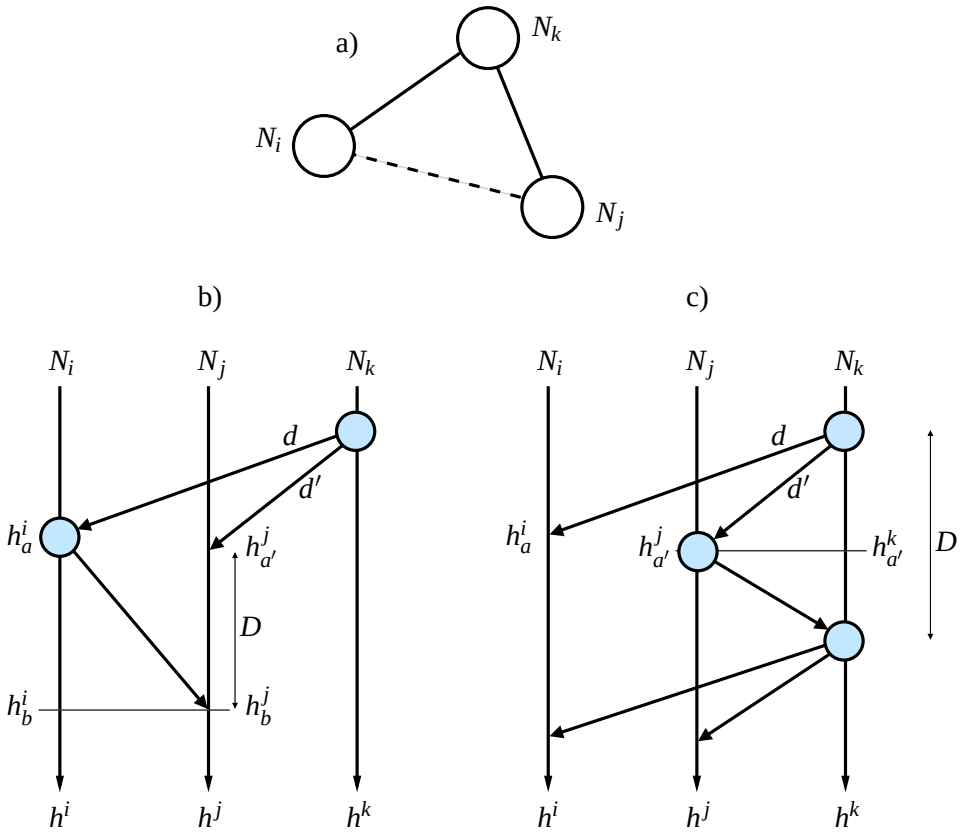

Fig. 5: *Reference-broadcast synchronization.* Node N_i determines the offset of its local clock relative to that of node N_j with the help of node N_k. In (c), a variant of reference-broadcast synchronization is shown that can be used if N_i and N_j cannot communicate directly with each other (dashed link in (a)).

The disadvantage of the reference-broadcast approach is that physical broadcasts and a beacon node are required. In large networks, this leads to clusters of nodes that are within broadcast range of each other. The maintenance of clusters and the synchronization between them creates additional overhead.

2.5.2 Synchronization in rounds

Typically, two local clocks do not run at exactly the same speed. Therefore their synchronization has to be refreshed periodically; the length of a synchronization round is determined by the maximal permissible synchronization error and by the relative drift between the two clocks. Let the length of a round be τ_{round}. Assume a round consists of a first period with length τ_{sample}, where one or more samples are taken according to one of the methods described in Section 2.5.1, and a second period where the nodes do nothing. Let us assume that an application allows for a total error of E_{total}, the maximum error after taking the samples is E_{sample}, and the maximal drift rate is $\hat{\rho}$. Then the length of a round τ_{round} has

to satisfy

$$\tau_{\text{round}} \leq \frac{E_{\text{total}} - E_{\text{sample}}}{\hat{\rho}} \quad .$$

The above relation implies that rounds can be longer if E_{sample} and $\hat{\rho}$ are small. For example, algorithms that use the round-trip technique can bound E_{sample} according to the measured round-trip time and thus can dynamically increase τ_{round} if the round-trip time was small. Other algorithms compensate for the drift of the local clock and therefore can compute a smaller effective $\hat{\rho}$, which also allows to increase τ_{round}.

In some applications, E_{total} is smaller than what can be guaranteed by taking a single sample. In such a case, multiple samples can be taken to achieve $E_{\text{sample}} < E_{\text{total}}$. Taking multiple samples increases τ_{sample}. At the limit, $\tau_{\text{sample}} \approx \tau_{\text{round}}$; in this case, synchronization in rounds becomes a continuous process.

2.5.3 Combining multiple time estimates

We now discuss techniques for combining multiple estimates of the local time of a remote node. Figure 6 (a) illustrates the situation: Every circle stands for a single estimate of node N^j's local time h_a^j at some event a, which occurs at N_i's local time h_a^i.

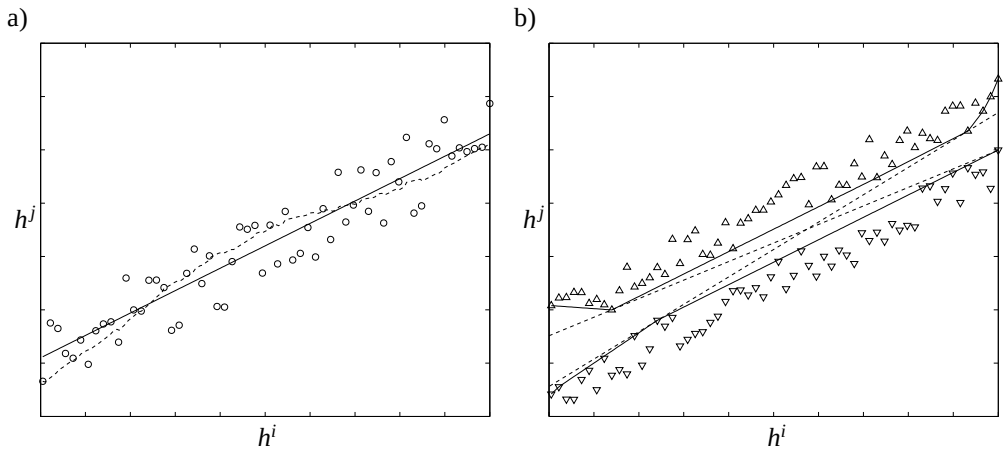

Fig. 6: *Multiple samples improve on the synchronization error.* (a) Every circle represents a sample, i.e. a local time h^i of node N_i and an estimated local time h^j of node N_j. Interpolation techniques can be used to reduce the synchronization error: The solid line results from a linear regression on the samples, the dashed line is the result of a phase-locked loop (PLL). (b) The same idea can be used for lower ($\bigtriangledown$) and upper ($\bigtriangleup$) bounds on the local time of N_j. Also here, interpolation can considerably reduce the synchronization error (i.e., the uncertainty in this case). The solid lines are determined by the convex-hull approach, the dashed lines according to [SV03].

Linear regression

The most widely used technique is linear regression, where a linear relation

$h^j = \alpha + \beta \cdot h^i$ is postulated and the coefficients α and β are determined by minimizing the square of the difference between the fitted h^j's and the actual samples. This technique has a single parameter, i.e. the number of samples that are accounted for when computing the coefficients. A large number of samples can improve the regression quality, but requires a large amount of memory.

The coefficient β can be interpreted as an estimation of h^j's drift relative to h^i. Linear regression thus implicitly compensates for clock drift. If the drift is variable, the postulated linear relationship between h^j and h^i does not describe reality very well. In such a situation, the number of samples accounted for should be small.

The linear regression can be computed online, i.e. incrementally whenever a sample is taken. An efficient implementation can be found in [PTVF92]. A disadvantage of the linear-regression technique is that it weighs data points by the square of their error against the fitted line. Outliers thus have a particularly strong influence on the resulting coefficients α and β.

Phase-locked loops
Another method for processing a continuous sequence of samples is based on the principle of phase-locked loops (PLLs) [Gar79]. The PLL controls the slope of the interpolation using a proportional-integral (PI) controller. The output of a PI controller is the sum of a component that is proportional to the input and a component that is proportional to the integral of the input. The input of the controller is the difference between the actual sample and the interpolated value. If the interpolation is smaller than the sample, its slope is increased, otherwise it is decreased. The main advantage of the PLL-based approach is that it requires far less memory than the linear-regression technique (in essence only the current state of the integrator sum). The main disadvantage is that PLLs require a long convergence time to achieve a stable rate [Nor00]. The NTP algorithm uses a PLL [Mil95].

Local selection
In [BT02], the local selection algorithm was introduced. It assumes unidirectional communication of time stamps from a synchronization master to a client. The client updates its synchronized time to the time contained in the time stamp if and only if the received time is larger than the client's current synchronized time. Up to here, the algorithm is identical to the one presented in [Lam78]. The distinguishing characteristic of the local selection algorithm is that the client updates its synchronized time assuming that its local clock is running at maximum speed. Thus, the client's synchronized time can never be larger than the master's reference time.

The algorithm compensates for the effect of large message delays: the time stamp of a message with a large delay is implicitly ignored, as the client's synchronized time will already have grown beyond the time contained in the time stamp. The maximal synchronization error is directly limited by the maximal message delay and the period between consecutive time stamps.

2.5.4 Combining multiple time intervals

The techniques of Section 2.5.1 can also be used to derive lower and upper bounds on the local time of a remote node. Figure 6 (b) shows a sequence of lower and upper bounds on the local times h^j of a remote node N_j on the y-axis and the corresponding local times h^i of a node N_i on the x-axis. In the previous section, the samples formed a single cloud and the interpolation was a line "through the middle of this cloud". Here, we have two clouds: one formed by the lower-bound samples, the other by the upper-bound samples.

The convex-hull technique [Ber00, ZLX02] interpolates the two clouds separately. One curve is drawn above all lower bounds, a second curve below all upper bounds. While linear-regression and PLL techniques tend towards the average of the individual samples, the convex-hull technique ignores average values and accounts for the samples with minimal or maximal error. This can result in improved robustness: While the current average message delay can be very unstable, the minimal delay remains stable, though it may occur rarely.

In [SV03], it is proposed to interpolate lower- and upper-bound samples by a single line as follows: First the steepest and flattest lines that do not violate any lower or upper bound are determined. The slopes of these lines represent bounds on the drift of clock h^j relative to h^i. The average of these two extremal solutions is used as the final interpolation; for a more detailed description, see page 39.

2.5.5 Synchronization of multiple nodes

Sensor networks most often have a much more complicated topology than the simple examples shown in Figures 4 and 5, and not all sensor nodes can communicate with each other directly. Thus, multi-hop synchronization is required, which adds an additional layer of complexity. Clearly, this could be avoided by using an overlay network which provides virtual single-hop communication from every sensor node to a single master node. But as we have seen in Section 2.5.1, the synchronization error directly depends on the message delay, which is very difficult to control on a logical link composed of many physical hops. Therefore, well-performing synchronization schemes have to deal with the multi-hop problem explicitly.

Figure 7 illustrates various approaches to multi-hop synchronization. We now describe these four schemes and use them as examples to discuss the main problems of multi-hop synchronization.

Out-of-band synchronization

The conceptionally simplest solution is to avoid the problem: A large number of master nodes is distributed in the network such that every node has a direct connection to at least one of these masters (e.g., [VRC97]). The master nodes are synchronized among each other using some out-of-band mechanism. GPS is well suited to this purpose as it provides time information with sub-microsecond accuracy. However, GPS receivers are still relatively costly, consume a considerable amount of energy, and require a free line of sight to a number of satellites.

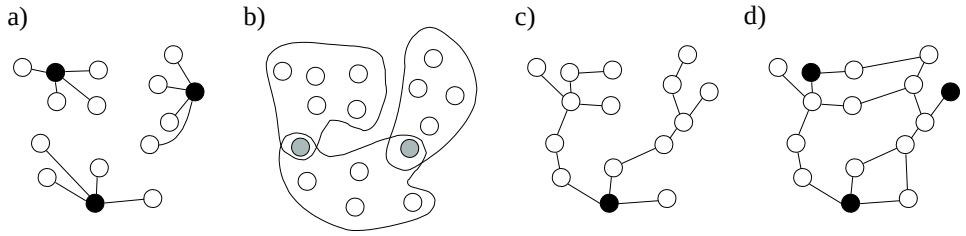

Fig. 7: *Organizing synchronization in multi-hop networks.* a) Single-hop with three master nodes that are synchronized out of band (e.g., using GPS). b) Single-hop synchronization in overlapping clusters with two gateway nodes that translate time stamps. c) Tree hierarchy with a single master node at the root. d) Unstructured.

Clustering

The authors of the RBS algorithm propose to partition the network into clusters [EGE02]. All nodes within a cluster can broadcast messages to all other members of the cluster and thus the reference-broadcast technique can be used to synchronize the cluster internally. Some nodes are members of several clusters and participate independently in all corresponding synchronization procedures. These nodes act as gateways by translating time stamps from one cluster to the other. There is a trade-off in choosing the size of the clusters. On the one hand, a small number of large clusters reduces the number of translations and thus reduces the synchronization error. On the other hand, energy consumption grows quickly with increasing transmission range; this makes choosing many small clusters attractive. This trade-off has been examined in [MR03].

Tree construction

The most common solution to the multi-hop synchronization problem is to construct a synchronization tree with a single master at the root and to apply single-hop synchronization along the tree edges (e.g., [GKS03, SV03, vGR03, MKSL04]). Various well-known algorithms can be used to construct such a tree [vGR03]. As the error grows with the hop distance to the root, a tree with minimum depth is preferable. On the other hand, a small depth implies that the root has to serve many clients, and thus consumes far more energy than the other nodes.

Tree construction faces two main problems. Firstly, in sensor networks, the network topology may be dynamic: nodes may be mobile and repeatedly join or leave the network. The multi-hop synchronization algorithms have to explicitly deal with such events. In particular, if the root node fails, a new root has to be elected [MKSL04]. Secondly, two nodes that are physically close may have a large hop distance in the synchronization tree. In consequence, the quality of synchronization between these nodes is not as good as if they would synchronize directly with each other, and the fused data regarding an event observed by both nodes will be of lower quality.

Unstructured

As illustrated in the tree-construction approach, the multi-hop synchronization

problem can be seen as the problem of determining the links and directions over which time information is disseminated. In contrast to tree-construction approaches, unstructured approaches do not first explicitly solve this problem and then perform pairwise synchronization. Instead, time information is exchanged between any pair (or group) of nodes that communicate. Whereas in the tree-construction approach every pairwise synchronization is asymmetrical (i.e., between a local master and a client), in the unstructured approach it is symmetrical (i.e., between two equal peers). In Chapter 3, we present such an approach for interval-based synchronization: Two nodes combine their bounds on real time by selecting the larger lower bound and the smaller upper bound. A similar approach for single-value estimates is *asynchronous diffusion* proposed in [LR04]. Here, nodes that communicate adjust their synchronized clocks to the average of their synchronized times. Both approaches are completely local; as they do not maintain any global configuration, node mobility does not cause particular problems. In contrast, clustering and tree-construction schemes require the global configuration to be updated whenever nodes move or fail, or when new nodes are added to the system.

As algorithms that follow the unstructured approach do not attempt to communicate with a particular node (e.g., the parent node in a tree), some of these algorithms piggyback time stamps on messages that are sent for some other, not synchronization-related reason (e.g., [Röm01, BMT04]). It could be argued that these algorithms have virtually no communication overhead, as no messages are generated exclusively for time synchronization.

2.6 Case Studies

In the following, we discuss a number of concrete synchronization algorithms, ordered by publication date. Our goal is to give an overview of the approaches with reference to the techniques and classes discussed earlier in this chapter, rather than to discuss all the details. In addition, we will give some experimental results for each algorithm. Table 1 summarizes the underlying assumptions and classifies the approaches according to the criteria discussed in Section 2.4.

Time-Stamp Synchronization (TSS)

TSS [Röm01] provides internal synchronization on demand. Node clocks run unsynchronized, i.e. time stamps are valid only in the node that generated them. However, when a time stamp is sent to another node as part of a message, the time stamp is transformed to the timescale of the receiver. For messages sent over multiple hops, the transformation is performed for each hop.

Time-stamp transformation is achieved by determining the age of each time stamp from its creation to its arrival at a node. On a multi-hop path, the age is updated at each hop. The time stamp can then be transformed to the receiver's

	RBS	TPSN	TS/MS	LTS	TSS	IBS	TSync	FTSP	TDP	AD
Classes										
Internal vs. **E**xternal	I	E	I	E	I	E	E	I	I	I
Continuous vs. **O**n-demand	O	C	C	O	O	C	C	C	C	C
All nodes vs. **S**ubsets	S	A	S	A/S	S	A	A	A	A	A
Rate vs. **O**ffset	RO	O	RO	O	O	O	O	RO	O	O
Transform vs. **C**lock sync.	T	C	–	C	T	C	C	C	C	C
Ins**t**ants vs. In**t**ervals	S	S	TS	S	T	T	S	S	S	S
Assumptions										
Broadcast	X	X					X	X	X	X
Bidirectional communication		X	X	X	X		X		X	X
Constant rate			X							
Bounded drift				X	X	X				
Multichannel							X			
MAC access		X						X		

Tab. 1: Synchronization classes and assumptions of time-synchronization protocols.

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 transfer the time stamp from node to node. The first component is measured using the local, unsynchronized clocks. The second component can be bounded using the round-trip times of the messages and their acknowledgments.

For the round-trip measurement, the technique depicted in Figure 4 (d) on page 30 is used, where the sender is N_i and the receiver is N_j. Message d_2 is a data message containing the time stamp, message d_2' is an acknowledgment. Using the previous message exchange (d_1, d_1'), the receiver can use $D^j - D^i$ as an upper bound on the delay of message d_2. If a minimum delay is known, it can be used as a lower bound; otherwise, 0 is used. The storage time and the above bounds on transmission delay can be used to determine lower and upper bounds on the time-stamp age. Additionally, $\hat{\rho}$ is used to transform time intervals between node clocks as in (2.2) on page 23.

With this approach, synchronization information is piggybacked on existing (acknowledged) messages. There are no additional synchronization messages, except when two nodes exchange a message for the first time. In this case, an additional initialization message must be sent and acknowledged in order to enable round-trip measurement. An acknowledgment is not needed if the sender can overhear the receiver forwarding the message over the next hop, which is typically the case in broadcast networks.

Measurements in a wired network with $\hat{\rho} = 1$ ppm showed that the average uncertainty of the time-stamp interval is about 200 µs for adjacent nodes. It increases by an additional 200 µs for each additional hop, and by about 2.5 µs per age second.

Reference-Broadcast Synchronization (RBS)

RBS [EGE02] provides synchronization for a whole network. The basic synchronization primitive is a reference broadcast to a set of client nodes in the one-hop neighborhood of a beacon node as illustrated in Figure 5 (b) on page

32. The beacon broadcasts synchronization pulses; the clients exchange their respective reception times and use linear regression to compute relative offsets and rate differences to each other. Using offset and rate difference, each client can transform a local clock reading to the local timescale of any other client.

To extend this scheme to multi-hop networks, the network is clustered such that all nodes in a cluster can be synchronized by a single beacon node. Gateway nodes are members of two or more clusters and independently participate in the reference-broadcast procedures of all their clusters. By knowing offsets and rate differences to nodes in all adjacent clusters, gateway nodes can transform time stamps from one cluster to another. Time synchronization across multiple hops is then provided as follows: Nodes time-stamp sensor data using their local clocks. Whenever time stamps are exchanged among nodes, they are transformed to the receiver's local time using offset and rate difference.

Experiments have shown that adjacent Berkeley Motes [HC02] can be synchronized with an average error of 11 µs by using 30 broadcasts. Over multiple hops, the average error grows with $\sqrt{n}$, where n is the number of hops.

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

Tiny-Sync and Mini-Sync [SV03] 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. A constant-rate model (see page 23) is assumed. To obtain data points for line fitting, multiple round-trip synchronizations are performed as depicted in Figure 4 (c) on page 30, where the client is N_j and the reference is N_i. Each round-trip measurement results in a data point $(h_b^i, [h_a^j, h_c^j])$. Then, the line-fitting technique depicted in Figure 6 (b) on page 33 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 could be 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, and Mini-Sync gives an optimal solution with increased overhead.

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

Lightweight Time Synchronization (LTS)

LTS [vGR03] provides a specified precision with little overhead, rather than striving for maximum precision. Two algorithms are proposed: one that operates on demand only for nodes that 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 first constructs spanning trees (with the masters at the roots) by flooding the network. In a second phase, each node synchronizes with its parent 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 $\hat{\rho}$.

In the on-demand version, a node that needs synchronization sends a request to one of the masters using some (not further specified) routing algorithm. 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 requests. If there are any such requests, the node synchronizes with the neighbor, rather than executing a 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 h. 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 [GKS03] provides synchronization for a whole network. First, a node is elected as a synchronization master, and a spanning tree with the master at the root is constructed by flooding the network. In a second phase, each node performs round-trip synchronization with its parent in the tree; this synchronization is done 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 after their parents have synchronized. To minimize message-delay uncertainties, time-stamping for the round-trip synchronization 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 [EGE02]. However, the authors of [GKS03] 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 [DH04] 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, one or more master nodes with access to a reference time are assumed to exist.

HRTS constructs a spanning tree with the master at the root. The master uses the reference broadcasting technique illustrated in Figure 5 (c) on page 32 to synchronize its children, and each child node then 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)

In [BMT04, MBT04], interval-based synchronization was proposed as particularly suited for sensor networks. This is also the main idea of this thesis and will be discussed in detail in Chapter 3.

The use of guaranteed bounds was first proposed in [MO83] for a bounded-drift model (see page 23). 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 time and the known drift bounds. An analysis of the best achievable synchronization was provided in [PSR94]. In [AHR96], different delay models were examined, while clocks were assumed to be perfect. In [SS97], the model was refined by including bounded drift variation and fault-tolerance.

Flooding Time-Synchronization Protocol (FTSP)

FTSP [MKSL04] can be used to synchronize a whole network. Each node is assigned a unique ID; the node with the lowest ID is elected as a leader that serves as a source of reference time. If this node fails, then the remaining node with the lowest ID is elected as the new leader. The leader periodically floods the network with a synchronization message that contains the leader's current time. Nodes which have not received this message yet record the contained time stamp and the time of arrival, and broadcast the message to their neighbors after updating the time stamp. Each node collects eight (time stamp, time of arrival) pairs and uses linear regression on these eight data points to estimate offset and

rate difference to the leader. Time-stamping is performed in the MAC layer to minimize the delay uncertainty.

Measurements were performed in an 8-by-8 grid of Berkeley Motes, where each Mote has a radio link to its eight closest neighbors. With this setup, the network synchronized in 10 min 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 [LR04] provides 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. This sequence of operations is assumed to be atomic, i.e. 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 Protocol (TDP)

TDP [SA05] provides internal or external synchronization of a whole network. Initially, a set of leader nodes is elected. For external synchronization, these nodes must have access to a global time. This is not required for internal synchronization, where leaders are initially unsynchronized.

Leader nodes then broadcast a request message containing their current time, and each node that receives this message replies. Using round-trip measurements, each leader node calculates and broadcasts the average message delay and standard deviation. Non-leader nodes record these data for all leaders they are receiving messages from. Then, they 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 leaders. The diffusion procedure stops at a given number of hops from the leaders.

For each leader l that a node has received data from, the node also possesses the time h_l at the initial leader, the accumulated message delay Δ_l, and the accumulated standard deviation β_l. The node computes a clock estimate as $\sum_l \omega_l(h_l + \Delta_l)$, where the weights ω_l are inversely proportional to the standard deviation β_l. After all nodes have updated their clocks, new masters are elected and the procedure is repeated until all clocks have converged to a common time.

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

2.7 Evaluation Strategies

Evaluating the precision of time synchronization in wireless sensor networks is not a trivial task. For example, the authors of the RBS algorithm report 11 µs precision on the Berkeley Motes platform [EGE02], while the authors of the TPSN algorithm report 29 µs for RBS on the same platform, concluding that TPSN is better, as it achieves 17 µs [GKS03].

In this section, we discuss different evaluation strategies that have been applied to time-synchronization algorithms for wireless sensor networks. There are various aspects of the performance achieved by an algorithm that can be evaluated, for example the energy consumption or the message and memory overhead. The discussion in this section concentrates on various alternatives for the evaluation of the *precision* of time-synchronization algorithms.

2.7.1 What is precision?

Figure 2 (b) on page 27 illustrates the scope and lifetime of synchronization in a sensor network. The scope defines which nodes have to be synchronized and the lifetime defines at which time these nodes have to be synchronized. Thus, it is natural to evaluate the precision in the shaded area of Figure 2 (b). The precision is a metric that is closely related to the synchronization error. While the precision is a single scalar value for a whole network, the synchronization error is a function of time for a single node. In the following, we discuss several alternatives to map such functions to a single scalar precision value P.

Combining the synchronization errors of many nodes

At some time t within the lifetime of a sensor network, every node N_i within the scope has a synchronized time $c^i(h^i(t))$. In the case of internal synchronization, the *instantaneous precision* $p(t)$ is often defined as the maximal difference between any two synchronized times, i.e.

$$p(t) = \max_{i,j} \left\{ c^i(h^i(t)) - c^j(h^j(t)) \right\} \quad .$$

Some authors (e.g., [SA05]) use the standard deviation among all $c^i(h^i(t))$ as a measure for the instantaneous precision at time t.

In the case of external synchronization, the instantaneous precision is more often defined as the maximal synchronization error, i.e.

$$p(t) = \max_{i} \left\{ c^i(h^i(t)) - t \right\} \quad .$$

This variant of precision is sometimes called *accuracy*. Alternatively, the precision can be defined as the average synchronization error within the scope or the maximal synchronization error among a given percentage of the nodes in the scope with the smallest synchronization error.

Note that obviously the common definitions of precision and accuracy described above clash with the usual meaning of these words. It would be more correct to speak of imprecision and inaccuracy.

Steady state and convergence time

The instantaneous precision $p(t)$ obviously varies during the synchronization lifetime. The final precision metric P can be defined as the maximum of $p(t)$ for all t in the lifetime. Alternatively, the average of $p(t)$ can be used.

The precision P improves in proportion to the time the synchronization process is active, and at some point, the improvement stops. Usually, the precision P is evaluated after this point, i.e. the lifetime of synchronization starts after the synchronization process, and the precision P describes the steady state.

Some authors consider the *convergence time*, which is the length of the interval from the start of the synchronization process to the point in time where the precision P stops improving or reaches a specific value. If the lifetime is defined, the convergence time indicates when the synchronization process has to be started such that the desired precision P is achieved before the start of the lifetime and is maintained until the end of the lifetime.

2.7.2 Goals of performance evaluation

There can be different reasons why the performance of an algorithm has to be evaluated, all leading to different solutions.

The actual performance of a given synchronization algorithm strongly depends on the properties of the target platform. It is difficult to identify and model all the influencing factors explicitly. A realistic estimation of the achievable precision is thus best obtained by measurements on the actual target platform, rather than by simulation of a simplified target platform.

Sometimes, realistic estimation of the performance is less important than fairness and repeatability of the evaluation. This is the case if several competing algorithms have to be compared; it is important that differences in the performance are due to differences in the algorithm and not due to different conditions (e.g., message delays, clock drifts). Here, simulation based on recorded or generated traces is more appropriate than direct measurements.

If the goal of analyzing a particular synchronization algorithm is to give worst-case guarantees on its performance, neither measurements nor simulation based on recorded traces can be used, since both strategies only evaluate a finite number of instances. Instead, the worst-case has to be identified and the worst-case performance has to be determined analytically.

2.7.3 Measurements

Measurement techniques

Three fundamentally different measurement strategies, which are illustrated in Figure 8, have been used in recent publications.

Consider Figure 8 (a). Every sensor node executes two synchronization procedures, synchronizing two different clocks. The first procedure is the actual synchronization algorithm under test, using only the means of the platform on which it is executed. The second procedure is another algorithm which achieves

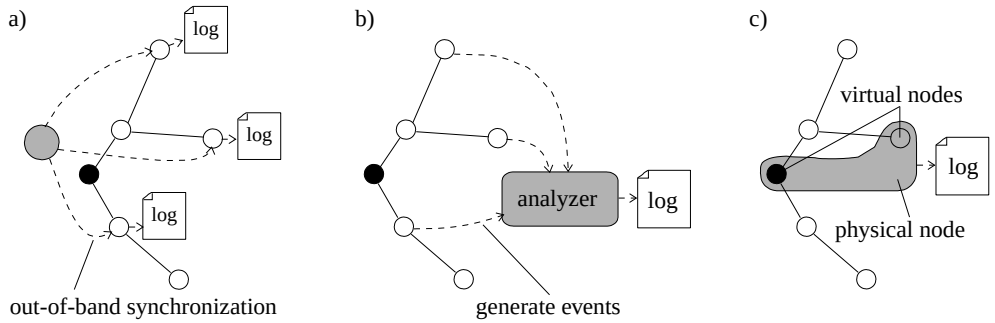

Fig. 8: *Precision-measurement techniques.* a) Every node is synchronized out of band and measures its own precision. b) Every node generates events, the evaluation is centralized. c) The master node and a client node are virtual nodes on the same hardware device.

a far better precision than the first; this is made possible by additional resources that are not offered by the target platform, but which are introduced for the measurements. For instance, a GPS receiver for every sensor node can provide these resources. Alternatively, cable connections can be used as an out-of-band mechanism with very low delay variability to provide a reference time (e.g., [MFNT00, EGE02, BT02, MFT05]). In [MKSL04], a single-hop RBS scheme is used to measure the precision achieved by the FTSP multi-hop algorithm. This approach has the advantage that every node can evaluate and log its own precision and these values can be collected at the end of the experiment (or even online), providing complete information.

An alternative is shown in Figure 8 (b). All sensor nodes generate some directly observable event, for example a rising edge on an I/O pin, when their synchronized time reaches a particular value X. An external analyzer device then records the time interval between the instance when a node's synchronized time is X and the instance when the real time is X. Such a procedure was used for example in [GKS03]. Its advantage is that the precision of the measurement is not limited by the resolution of the nodes' clocks or by the performance of a second synchronization procedure.

As illustrated in Figure 8 (c), [Röm01] proposes to measure the precision achieved by *one* client node as follows: A client node synchronizes over several hops to a master node. Master and client nodes are virtual nodes emulated on a single physical node, the intermediate nodes are all separate physical nodes. As the master and the client share a single hardware clock, the precision of the client can easily be evaluated.

Systems and topologies

All three approaches do not scale well; hence, measurements have been done only in small networks so far. The largest experiment is described in [MKSL04], where an 8-by-8 grid of Mica2 Motes is evaluated. In [GKS03], a chain of 6 Mica Motes is used, [DH04] evaluates 5 MANTIS Nymph nodes, [Röm01] evaluates a chain of 7 standard PCs with 100 Mbit/s Ethernet, and [EGE02] evaluates IPAQ nodes communicating via IEEE 802.11b WLAN and Mica Motes.

It is an open question how to measure the synchronization error of hundreds of nodes. Current evaluations of large networks are based on simulation.

Results

We will now give some measurement results from recent publications. Our intention is to give an idea about the order of magnitude of the achievable precision and to illustrate that although all results are about precision, they are difficult to compare.

In [MKSL04], the convergence time of the FTSP algorithm in an 8-by-8 grid is reported to be 10 min. A maximal error of 38 µs and an average error (over all nodes) of 12 µs is reported. For the TPSN algorithm, [GKS03] reports a maximal error of 45 µs for one hop and 74 µs for five hops. Average errors (over time) are 17 µs for one hop and 38 µs for five hops. The authors provide also the percentage of the time when the synchronization error was below the average error (> 60 %). In [EGE02], the authors of RBS present the distribution of the synchronization error (over time) for one hop and the mean, median, 95 %, and 99 % values over 300 trials for one to four hops.

Some authors evaluate the distribution of the synchronization quality in the system. At some time t, either the synchronized times $c^i(h^i(t))$ of all nodes i [SA05], or alternatively the corresponding synchronization errors $e^i(t)$ [GKS03, DH04], are shown in a histogram.

2.7.4 Simulation

Performance evaluation through simulation has the advantage that the resulting precision or accuracy of all nodes does not have to be measured but is directly accessible. Thus, much larger systems can be evaluated.

Systems and topologies

In [SA05], systems with 200 nodes are evaluated, in [LR04] and [vGR03] up to 500 nodes, always randomly placed in a square area. The transmission range of the nodes is 10 m in a square with edge length 80 m [SA05] or 120 m [vGR03]; in [LR04], various transmission ranges from 0.4 m to 1 m are used in a square with edge length 10 m. In [BMT04], the range is varied between 0.1 and 0.5 times the width of the square area. In [SV03], a chain of 5 nodes is simulated.

Message delays

For simulation, a number of assumptions about the behavior of the system have to be made. In [SV03], measured delay traces from an IEEE 802.11 wireless LAN are used, [SA05] and [vGR03] generate delay traces according to a normal distribution. In [SA05], an additional offset is added which increases when the medium is saturated, i.e. when more than 75 % of the channel capacity is used. The authors of [BMT04] assume zero message delay, arguing that the synchronization errors induced by delay uncertainty and drift can be studied separately. We will explore this in Chapter 3.

Clock drift

In [SA05] and [BMT04], every node is assigned a randomly chosen, constant

drift rate between -100 ppm and $+100$ ppm. In [vGR03], all nodes have a drift rate of 50 ppm.

Results

The main concern of [vGR03] is to compare centralized and distributed versions of the LTS algorithm in terms of required messages and achieved synchronization error. The average error (over all nodes) is evaluated as a function of the hop distance to the master node.

In [SV03], the synchronization and the drift-compensation error achieved by the TS/MS algorithms are evaluated as a function of time. A node one hop away from the master has an error of 1 ms after 83 min. A node with five hops distance achieves 3 ms.

In [BMT04], the average synchronization error over time and over all nodes is evaluated as a function of the number of messages exchanged between the nodes. Also the impact of the transmission range and of the number of master nodes is analyzed.

[LR04] mainly looks at how quickly a network synchronizes using the AD algorithm. The number of rounds is evaluated as a function of the transmission range and of the number of nodes in the system. It is also shown that the synchronization error decreases exponentially with the number of rounds.

The speed of convergence is also evaluated in [SA05], here for the TDP and TPSN algorithms; the standard deviation of the nodes' synchronization error is shown as a function of time. It is argued that node mobility makes convergence slower. In addition, histograms and three-dimensional plots of the distribution of the synchronization error after convergence are presented.

2.7.5 Challenges of a benchmark

So far, we have presented how synchronization algorithms are evaluated in current literature. We have seen that results of different authors are quite incomparable due to widely differing goals, assumptions, and techniques. On the one hand, there is not yet a common understanding about the requirements on synchronization in sensor networks. On the other hand, there is also disagreement about available resources and platforms.

A benchmark for comparing the various algorithms on common grounds has not yet been presented. It is difficult to devise a benchmark that can be used with a large number of algorithms: Ideally, the comparison of algorithms is based on simulation using system traces. Such traces should contain the system and communication model (How many nodes are there? How many of them are reference nodes? Which node communicates with which other node at which time?), and they should characterize the "adversary" of synchronization, namely all message delays and the drift rates of the nodes. But this would require to determine all communications before executing the algorithms. This is not possible for most of the algorithms, since they actively generate messages, depending on previous events. Furthermore, some algorithms require broadcast communication, while others do not.

2.8 Summary

In this chapter, we discussed various aspects of time synchronization in sensor networks. We outlined the applications of physical time and discussed why existing synchronization algorithms from the distributed-systems area typically cannot be used in wireless sensor networks. We also presented common classes of and techniques for synchronization, reviewed various time-synchronization algorithms proposed specifically for wireless sensor networks, and discussed evaluation strategies.

The case studies of time-synchronization algorithms and the discussion of evaluation techniques illustrated the very real problem of evaluating and comparing synchronization algorithms. Note that these difficulties do also apply to calibration and many other distributed algorithms. One of the challenges for future research is hence the development of methods and tools for the evaluation of time synchronization and calibration in large-scale sensor networks.

Current application-oriented projects (e.g., [JOW+02]) indicate that many simplifying assumptions about sensor networks (e.g., immobile nodes, fixed network topology) may not hold in practice. Hence, future work might have to revisit existing approaches for time synchronization under updated assumptions.

3

Interval-Based Synchronization

3.1 Introduction

The main idea of this thesis is to propose the use of time intervals given by guaranteed bounds as a particularly suited approach for time synchronization in wireless sensor networks. We believe that in such networks, guaranteed bounds offer distinct advantages over time estimates:

- Time-stamping single sensor-data items with guaranteed bounds allows to obtain guaranteed bounds from sensor-data fusion, such as guaranteed bounds on the speed of an object.
- Nodes are provided with a deterministic quality measure for their time information. In hard-real-time applications, a node can detect that its time uncertainty has grown excessively large and take appropriate action, such as request new time information or enter a fail-safe state.
- The concerted action (sensing, actuating, or communicating) of several nodes at a predetermined real time always succeeds: each node can minimize its uptime (and hence maximize its energy savings) while guaranteeing its activity at the predetermined real time. Time-division multiplexing may benefit from guaranteed bounds, e.g. via reduced guard times between successive time slots.
- The combination of intervals by intersection is unambiguous and optimal, while the reasonable combination of time estimates requires additional information about the quality of the estimates. As a consequence, interval-based synchronization naturally solves the problem of choosing reference nodes in large networks.

In Section 3.2, we present a new system model for the analysis of interval-based time synchronization in mobile ad-hoc networks. We justify why our abstractions are well chosen for this class of networks. Using our system model, we first consider external synchronization and derive worst-case bounds on the quality of interval-based synchronization in Section 3.3. More specifically, we provide lower bounds on the synchronization error for an arbitrary communication pattern. We show that a very simple algorithm from [MO83] is worst-case-optimal. However, this algorithm is not optimal in the average case. In Section 3.4, we present an algorithm that is also worst-case-optimal but achieves better synchronization quality in the average case than the algorithm from [MO83].

We shift our attention to internal synchronization in Section 3.5. We use our model to analyze and improve an algorithm from [Röm01], and then present an algorithm that achieves optimal internal synchronization, albeit at the cost of high memory and communication overhead, in Section 3.6. We present an analogous algorithm for external synchronization in Section 3.7 and describe how limiting the amount of data that is stored and communicated affects the running time and the synchronization quality.

In Section 3.8, we show that interval-based synchronization does not need particular communication patterns such as trees or clustered hierarchies. This makes the interval-based approach resilient to node mobility, as there are no broken topologies that have to be repaired. Our simulation results suggest that mobility actually improves interval-based synchronization by increasing the rate of information dissemination through the network.

This chapter is partly based on [BMT04, MBT04, MBT05].

3.2 New Model for Time Synchronization

We propose a model for time synchronization in mobile ad-hoc networks; it allows us to identify the worst and the best case in terms of achievable time uncertainty and to analyze and improve existing synchronization algorithms. We will first make the case for our model's key element: the elimination of message delays. We then specify our event-based network model.

3.2.1 Zero-delay model

We now introduce and justify the key idea of our model, the elimination of message delays.

Delay uncertainty vs. drift

Clock-synchronization algorithms face two problems: The information a node has about the local time of another node degrades over time due to clock drift (the two clocks "drift apart"; this is illustrated in Figures 9 and 10), and its improvement through communication is hindered by message-delay uncertainty.

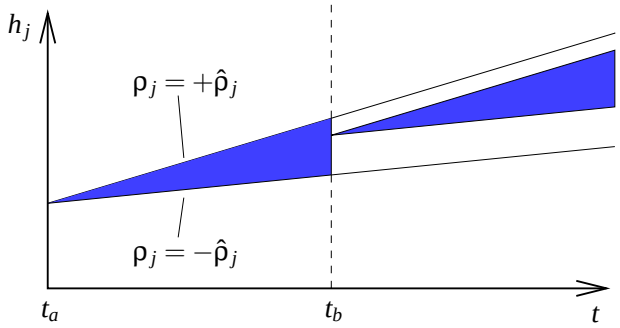

Fig. 9: Graphical representation of the knowledge of node N_i about the local time h^j of node N_j as a function of real time t. The shaded area is the region in which h^j can lie. The two nodes communicate at events a and b.

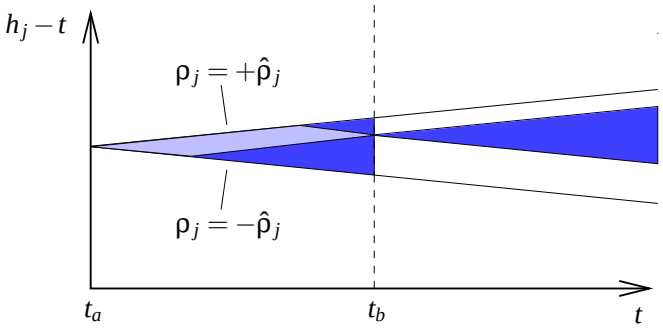

Fig. 10: Here, $h^j - t$ is plotted against t; additionally, the information about past values of h^j that N_i gathers at event b is shown as a lighter-shaded area.

These effects of delay uncertainty and drift are independent and can hence be studied separately.

We claim that in mobile ad-hoc networks, the influence of the clock drift typically dominates over that of the message delays. The reason for this is that communication is infrequent.

Infrequent, application-driven communication

Most of the work on time synchronization in ad-hoc networks has concentrated on the delay uncertainty. Recently proposed algorithms reduce it to a few microseconds (e.g., [BT02, EGE02, HC02, MFNT00]) and then achieve good synchronization by continued and frequent communication, which keeps the impact of clock drift negligible. But frequent and reliable communication is not always possible in mobile ad-hoc networks where energy is scarce and nodes may fail or be temporarily out of reach.

As the frequency of communication decreases, the uncertainty due to clock drift increases, while the uncertainty due to message delays remains constant. A numerical example: If the delay uncertainty is 1 µs and the clock drift's absolute value is bounded by 100 ppm, then after 5 ms, the drift's contribution to the uncertainty equals that of the delay. After one hour, it is 720000 times larger.

Even for "optimistic" values of 1 ms (uncertainty) and 10 ppm (drift), drift and uncertainty have equal impact after 50 s.

Moreover, due to the scarcity of energy, the synchronization service should only use the communication that takes place anyway for achieving the overall goal of the network: the time information is sent piggyback with the application data [PSR94, AHR96]. Therefore, the communication pattern is typically not known to the synchronization algorithm. Examples of such networks are sensor networks in which environmental data is collected on a regular basis but communicated only sporadically, e.g. when time-critical data is recorded, when a master node explicitly requests the data, or when the solar cells of the node are providing sufficient energy for communication.

Zero-delay model

We argued that the drift's impact on the time uncertainty is dominant in typical mobile ad-hoc networks. It hence has to be taken into account explicitly, while neglecting the delay uncertainty becomes more and more acceptable as the communication frequency decreases. Apart from this, time-stamping on relatively simple devices such as sensor nodes can be done at a low level (e.g. in the MAC layer), which leads to small uncertainties. Some algorithms reduce the uncertainty to a few microseconds, e.g. by using packet streams[1] (where a few packets will attain the minimal delay) [BT02] or reference broadcasts (which arrive almost simultaneously at different receivers) [EGE02].

We therefore assume delay uncertainties to be negligible, and eliminate the delays themselves from our analysis, assuming communication to occur in zero time. Note that our model can obviously be extended to take delays and delay uncertainties into account; delay uncertainty and drift are orthogonal issues, and our model concentrates on the issue that is dominant in mobile ad-hoc networks.

3.2.2 Event-based network model

We assume that the network consists of a large number of *sensor nodes* which have a drifting local clock, and a considerably smaller number of *anchor nodes* that have access to time information of constant uncertainty[2]. We refer to nodes that are within each other's transmission range as *neighbors*.

To model the interaction of the nodes in the network, we define three types of events: source, communication, and destination events.

Def. 1: **(Source event)** *Any sensor node may receive bounds on the current real time by communicating with a neighboring anchor node. We model this as a source event s which occurs at the sensor node at real time t_s and provides an uncertainty of ΔT_s about the current real time.*

Def. 2: **(Communication event)** *Any two neighboring sensor nodes may exchange their time bounds by communicating with each other. We model this as a communica-*

[1] Sending many packets per communication is no contradiction to infrequent communication.

[2] E.g. via GPS where the uncertainty is smaller than 1 µs.

tion event c which occurs at both nodes at real time t_c. The nodes simultaneously acquire mutual knowledge about their time bounds.

This is a high-level abstraction. In reality, every communication takes a certain time, and may consist of a whole sequence of messages. In Section 3.4.4, we give quantitative evidence that this abstraction is well justified in infrequent-communication scenarios.

Def. 3: **(Destination event)** *A destination event d is an artificial event introduced for analysis purposes. A destination event can be defined for an arbitrary sensor node in the system and with an arbitrary real time t_d. In the analysis, we are then interested in the time bounds of that node at time t_d.*

The destination event is the time at which the node at which the event occurs reads its synchronized time, e.g. to time-stamp sensor data that was just acquired. Its name is due to the fact that the time information flows from the source event to the destination event, undergoing transformation on its way. This will become clearer in Section 3.3.2.

Def. 4: **(Scenario)** *We call the set of all events in a system, the real times at which they occur, and the time uncertainties associated with source events a* scenario.

A scenario describes a particular synchronization problem. For a given scenario, the local times of events and the bounds obtained at source events can still be chosen. If we fix these parameters, we obtain a *trace*:

Def. 5: **(Trace)** *We call the combination of a scenario, corresponding local times for all events, and the lower and upper time bounds*[3] *for the source events a* trace.

Def. 6: **(Admissible trace)** *We call a trace* admissible *if it satisfies the constraints on clock drifts as defined in (2.1) and the real times of source events are contained in the associated time bounds.*

3.3 External Synchronization

In this section, we use the model presented in Section 3.2 to identify the worst and the best case in terms of achievable time uncertainty, and to show the worst-case optimality of the algorithm IM from [MO83]. We then propose an improved algorithm which also is worst-case-optimal but yields better results in the average case. We show that this improvement is achieved by exploiting the typical drift diversity of the nodes' clocks. Simulation results show that the improvement is substantial for various sensor-network scenarios.

[3]The scenario contains the time uncertainty $\Delta T = T^u - T^l$, but not the bounds T^u and T^l.

In the following, we examine local, online algorithms which compute an interval in which the current *real time* is contained, i.e. algorithms that perform external synchronization. The interval is represented by a lower and an upper bound on real time. The goal of each network node is to minimize the size of this interval, i.e. the time uncertainty.

3.3.1 Worst-case analysis of the algorithm IM from [MO83]

In [MO83], an upper bound on the time uncertainty provided by the algorithm IM is given for completely connected networks with periodic communication. We extend this result to scenarios where connectedness and periodic communication are not guaranteed, thus making it applicable to mobile ad-hoc networks. We additionally show that the algorithm IM is worst-case-optimal by giving a lower bound on the uncertainty.

The algorithm IM from [MO83] is given below as Algorithm 1. During a communication event, nodes executing the algorithm exchange their current time bounds and intersect the intervals given by their own and by the received bounds. Specifically, each node calls the procedure *generateMessage* to obtain the time information to be incorporated in the outgoing message, and calls *processMessage* to evaluate the time information received.

Algorithm 1 Algorithm IM from [MO83]

procedure *initialize* [in: - / out: -]
  $(T^l_{\mathrm{M}}, T^u_{\mathrm{M}}) \leftarrow (-\infty, \infty)$ // most recent bounds
  $h_{\mathrm{M}} \leftarrow 0$ // local time when bounds were obtained

procedure *updateBounds* [in: $(T^l_{\mathrm{old}}, T^u_{\mathrm{old}})$,$\Delta h$ / out: (T^l, T^u)]
  $(T^l, T^u) \leftarrow \left(T^l_{\mathrm{old}} + \Delta h/(1+\hat{\rho}), T^u_{\mathrm{old}} + \Delta h/(1-\hat{\rho})\right)$

procedure *currentBounds* [in: h / out: (T^l, T^u)]
  $(T^l, T^u) \leftarrow \mathit{updateBounds}\left((T^l_{\mathrm{M}}, T^u_{\mathrm{M}}), h - h_{\mathrm{M}}\right)$

procedure *intersect* [in: $(T^l_A, T^u_A), (T^l_B, T^u_B)$ / out: (T^l, T^u)]
  $(T^l, T^u) \leftarrow \left(\max(T^l_A, T^l_B), \min(T^u_A, T^u_B)\right)$

procedure *generateMessage* [in: h / out: (T^l_c, T^u_c)] // at local time h
  $(T^l_c, T^u_c) \leftarrow \mathit{currentBounds}(h)$

procedure *processMessage* [in: $(T^l_c, T^u_c), h$ / out: -] // at local time h
  $(T^l_{\mathrm{M}}, T^u_{\mathrm{M}}) \leftarrow \mathit{intersect}\left((T^l_c, T^u_c), \mathit{currentBounds}(h)\right)$
  $h_{\mathrm{M}} \leftarrow h$

3.3.2 Path-based analysis

For a given scenario and a destination event d in this scenario, our goal is to compute the worst-case uncertainty $\Delta T_d = T_d^u - T_d^l$ provided by the algorithm IM. By the worst case we mean the most unfavourable, admissible trace for a given scenario. In the following, we consider the simple example shown in the event chart in Figure 11.

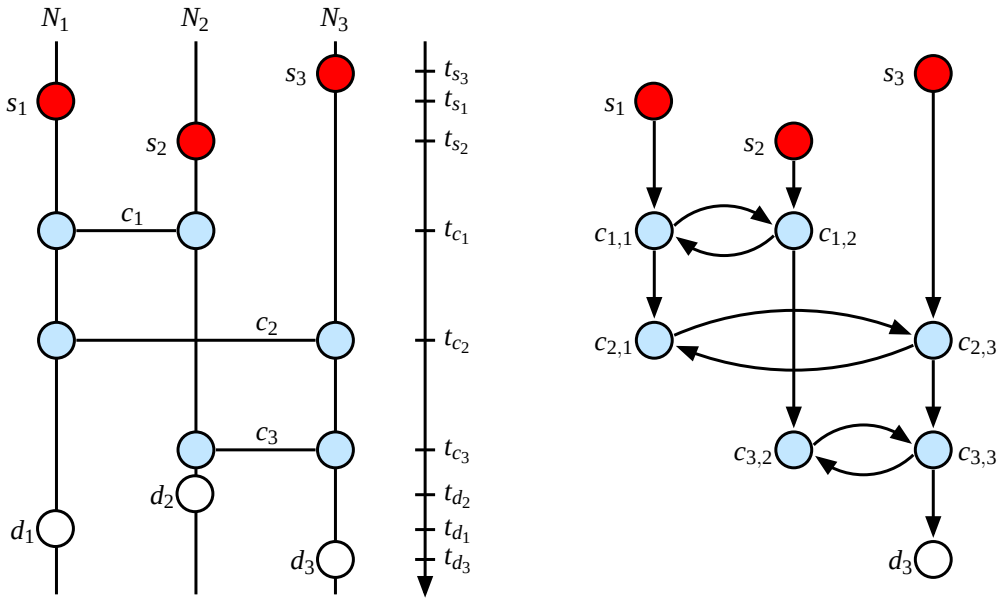

Fig. 11: The event chart on the left shows the times of all events and the involved nodes. The timing graph for destination event d_3 is shown on the right: Its vertices correspond to the circles that represent the events of the event chart, and its edges represent the information flow through the local states of the nodes.

Def. 7: **(View)** *For a given scenario and a real time t, the* view $V_i(t)$ *of node N_i is defined as all the information that node N_i could have obtained until time t. If knowledge about an event e is contained in $V_i(t)$, we write $e \in V_i(t)$.*

Def. 8: **(Timing graph)** *Let an event chart be given. For a destination event d in the event chart, the corresponding* timing graph *is obtained as follows:*

1. *For each non-communication event $e \in V_i(t_d)$, a vertex e is created. For each communication event $c \in V_i(t_d)$ which occurs at nodes N_i and N_j, two vertices c_i, c_j are created.*

2. *A directed edge (a,b) from a to b is created in the following cases:*

   (i) The vertices a and b are derived from two different events a, b which both happened at the same network node N such that a precedes b and there is no other event between the two. The weight of such an

edge (a,b) *represents the increase of uncertainty between the events a and b. It has the value*

$$\left(\frac{1}{1-\hat{\rho}} - \frac{1}{1+\hat{\rho}}\right)(h_b - h_a) \ .$$

(ii) The vertices a and b are derived from the same communication event. The weight of such an edge is 0.

Only the events which can have a causal effect on the given destination event d are contained in the timing graph; these are exactly the events $e \in V_i(t_d)$. The edges in the timing graph correspond to the transformation of the local-state information within a single network node between events, or to the information exchange at communication events. Note that for vertices a, b derived from the same communication event, the timing graph contains both edges (a,b) and (b,a). A path in the graph describes a causal dependency between the states of two network nodes. We call such a path a *communication path*. We will now see that the algorithm IM provides the uncertainty at a given destination event by computing a shortest path from some source event to the destination event.

Thm. 9: (Upper bound for the algorithm IM) *Let a network of nodes with bounded-drift clocks be given, and let these nodes employ Algorithm 1. For any destination event d occurring at time* t_d*, let* $\mathscr{S}$ *be the set of all source events in the corresponding timing graph. Then the time uncertainty* ΔT_d *can be bounded by*

$$\Delta T_d \leq \min_{s \in \mathscr{S}} \left\{ \Delta T_s + (t_d - t_s)\left(\frac{2\hat{\rho}}{1-\hat{\rho}}\right) \right\} \ .$$

Proof: We use a variant of the algorithm IM defined in Algorithm 1: we let *intersect* return (T_A^l, T_A^u) if $T_A^u - T_A^l < T_B^u - T_B^l$, and (T_B^l, T_B^u) otherwise. This algorithm clearly yields equal or larger values for the uncertainty ΔT_d than the algorithm IM. Therefore, an upper bound on its uncertainty is also an upper bound on the uncertainty of the algorithm IM. This variant is identical to the algorithm MM from [MO83].

We now assign to each vertex of the timing graph the uncertainty $\Delta T = T_{\mathrm{M}}^u - T_{\mathrm{M}}^l$ after the local-state update. The result of the modified algorithm IM can be interpreted as a shortest-path calculation on the timing graph (the original algorithm IM may use different paths for the lower and the upper bound). The uncertainty ΔT_d is bounded from above by the initial uncertainty ΔT_s at a source event s plus the length of a shortest path from s to d.

We now compute this length. From the clock model, we know that for any two events a and b, the inequality $h_b - h_a \leq (t_b - t_a)(1+\hat{\rho})$ holds. Calculating the path length in the timing graph gives

$$\Delta T_d \leq \Delta T_s + \left(\frac{1}{1-\hat{\rho}} - \frac{1}{1+\hat{\rho}}\right)(t_d - t_s)(1+\hat{\rho}) \ .$$

Some algebraic transformations yield the desired expression. ∎

3.3.3 Worst-case optimality

We now show that the algorithm IM is worst-case-optimal. We first give a universal lower bound on the uncertainty for a given communication scenario and destination event. This lower bound is equal to the upper bound from Theorem 9, hence the algorithm IM is worst-case-optimal. This was not shown in [MO83]. Our analysis naturally identifies the worst and best case for a given scenario.

Thm. 10: **(Lower bound)** *Let a communication scenario in a network of nodes with bounded-drift clocks and a destination event d be given. Let $\mathscr{S}$ be the set of all source events in the corresponding timing graph, and for all $s \in \mathscr{S}$, let ΔT_s be the uncertainty of source event s. Then there exists an admissible trace such that for any correct, deterministic and local clock-synchronization algorithm, the time uncertainty ΔT_d for the destination event d is bounded by*

$$\Delta T_d \geq \min_{s \in \mathscr{S}} \left\{ \Delta T_s + (t_d - t_s) \left(\frac{2\hat{\rho}}{1-\hat{\rho}} \right) \right\} \quad . \tag{3.1}$$

Proof: For simplicity, let us first assume that the time uncertainties at all source events are of size 0, i.e. $\Delta T_s = 0$ for all $s \in \mathscr{S}$. In this case, the minimum in (3.1) is attained by maximizing t_s, i.e. by choosing the latest source event $s \in \mathscr{S}$.

Proof for $\Delta T_s = 0,\ s \in \mathscr{S}$.
We derive the admissible trace by letting all clocks run with maximum speed. We now construct a second trace which is indistinguishable from the first, i.e. all views are identical in both traces. The real time at which the destination event d occurs is different in the two traces. The uncertainty of any correct, deterministic algorithm at event d has therefore to be at least as large as the time difference of event d in the two traces. This is illustrated in Figure 12: On the left, we have the trace where $\rho = +\hat{\rho}$ at all times. On the right, we have the constructed trace, in which from time $t_{\check{s}} = t_{s_1}$ on, all clocks run with minimal speed. Up to time $t_{\check{s}}$, both traces are identical.

For the two traces to be indistinguishable, all events must occur at the same *local* times, while the *real* times differ after time $t_{\check{s}}$. Note how the time intervals after time $t_{\check{s}}$ are stretched in the event chart on the right with respect to the chart on the left. Any event e with $t_e > t_{\check{s}}$ in the left-hand trace has to occur at such a time $\bar{t_e}$ in the right-hand trace that the local-clock time differences are identical in both traces. In the following, we will use $h_{\check{s}}$ to denote a local clock's reading at time $t_{\check{s}}$. From $h_e - h_{\check{s}} = (t_e - t_{\check{s}})(1+\hat{\rho}) = (\bar{t_e} - t_{\check{s}})(1-\hat{\rho})$, we obtain

$$\bar{t_e} = t_{\check{s}} + (t_e - t_{\check{s}}) \cdot \frac{1+\hat{\rho}}{1-\hat{\rho}}$$

and hence

$$\bar{t_e} - t_e = (t_e - t_{\check{s}}) \cdot \frac{1+\hat{\rho}}{1-\hat{\rho}} - (t_e - t_{\check{s}}) = (t_e - t_{\check{s}}) \left(\frac{2\hat{\rho}}{1-\hat{\rho}} \right) \quad ,$$

which for $e = d$ proves the claim for $\Delta T_s = 0$.

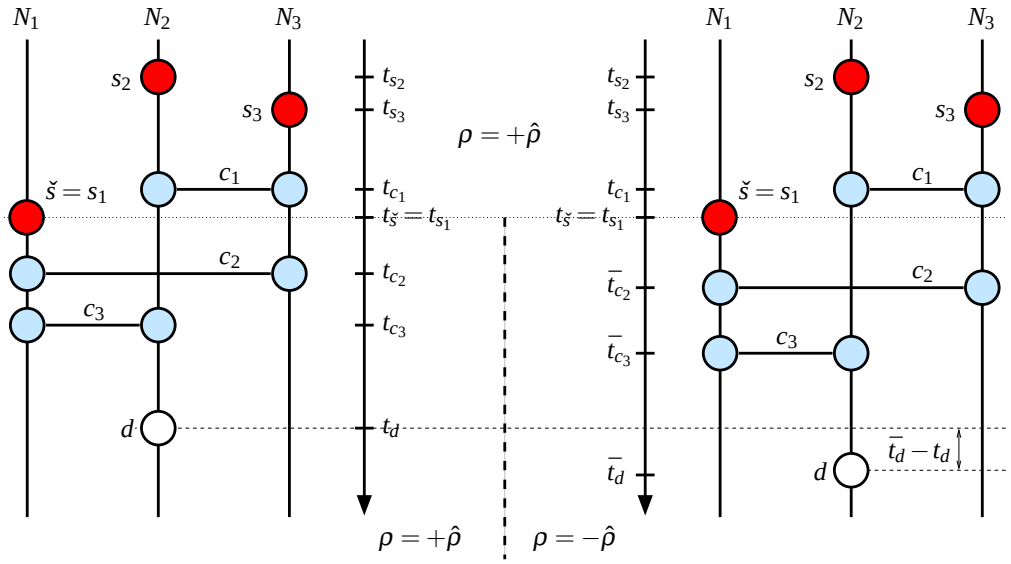

Fig. 12: Two indistinguishable traces. On the left, $\rho = +\hat{\rho}$ at all times. After time $t_{\check{s}}$, the clocks in the constructed trace on the right run with minimal speed. As can be seen, the event $\check{s}$ is the latest source event from which there is a path to d. As the two traces are indistinguishable from each other, the time uncertainty at event d has to be at least $\bar{t}_d - t_d$.

Proof for $\Delta T_s \geq 0,\ s \in \mathscr{S}$.
The proof is essentially the same as the one for $\Delta T_s = 0,\ s \in \mathscr{S}$; we only identify the differences in the following. To completely specify the traces, we have to give the bounds at source events. For a source event s with uncertainty ΔT_s, we set $T_s^l = t_s$ and $T_s^u = t_s + \Delta T_s$. Now, the event $\check{s}$ for which the minimum in (3.1) is attained is not necessarily the latest source event. We have to ensure that for all source events s with $t_s \geq t_{\check{s}}$, the time $\bar{t}_s$ at which event s occurs in the constructed trace lies within the time bounds of event s, i.e. that $T_s^l \leq \bar{t}_s \leq T_s^u$. To achieve this, we set $\bar{t}_s = \Delta T_{\check{s}} + t_{\check{s}} + (t_s - t_{\check{s}})\frac{1+\hat{\rho}}{1-\hat{\rho}}$. Note that in the constructed trace we therefore also have $\bar{t}_{\check{s}} = T_{\check{s}}^u$, and for any event e that occurs at time $t_e \geq t_{\check{s}}$ in the original trace, we obtain

$$\bar{t}_e - t_e = \Delta T_{\check{s}} + (t_e - t_{\check{s}})\left(\frac{2\hat{\rho}}{1-\hat{\rho}}\right) \quad ,$$

which for $e = d$ proves the claim for $\Delta T_s \geq 0$. ∎

Cor. 11: *The lower bound in Theorem 10 is tight.*

Cor. 12: *The algorithm IM is worst-case-optimal.*

Corollary 11 is a direct consequence of Theorem 9: The algorithm IM is guaranteed to achieve a time uncertainty which is not greater than the lower bound given in Theorem 10. Corollary 12 follows directly from Theorems 9 and 10 and Corollary 11.

Worst-case and best-case traces.
The proofs of Theorem 10 and Corollary 11 illustrate that the worst-case trace for a given scenario is the one where all clocks run with maximum speed. Analogously, the best case occurs when the drift difference of any two communicating nodes is maximal.[4] Then the intersection of the two time intervals at communication events is minimal.

3.4 Improved External Synchronization

In this section, we present and analyze an improved version of the algorithm IM, the Back-Path Interval Synchronization Algorithm (BP-ISA). The BP-ISA is worst-case-optimal like the algorithm IM, but achieves better results on non-worst-case traces.

3.4.1 Computing back paths

The BP-ISA is given below as Algorithm 2. The improvement to the algorithm IM consists in the following: For each neighbor, a node stores the time bounds of the last communication event with this neighbor. Whenever a node can improve its current bounds through communication, it tries to use them to also improve its stored bounds. At communication events, the nodes exchange their current bounds and the bounds of the previous encounter (if there has been one). Each node then uses both bounds to improve all bounds in its view.

The use of previous bounds introduces additional paths in the timing graph, as we will now explain. We showed that the algorithm IM computes shortest paths in the timing graph corresponding to a scenario. We now extend the timing graph by adding an edge (b,a) for each two vertices a and b that are derived from two different events which both happened at the same network node N such that a precedes b and there is no other event between the two. In the timing graph in Figure 11, each edge directed downwards is complemented by an inverse edge. The new edges may result in new and possibly shorter paths from a source event to a destination event. By considering these paths, the BP-ISA can achieve better time uncertainties than the algorithm IM.

3.4.2 Comparison of the algorithms

The BP-ISA has additional state information: The arrays $T_{\mathrm{M}}^{l}[N]$, $T_{\mathrm{M}}^{u}[N]$, and $h_{\mathrm{M}}[N]$ store the bounds on real time and the local time for the last communication event with every neighbor node.

The procedure *updateBounds* can also compute bounds for past events, i.e. for a negative Δh. Note the change of the sign in the terms $\Delta h/(1\pm\hat{\rho})$: While

[4] Clearly, this can occur only for very specific communication patterns. In Figure 11, not all three possible pairs of communicating nodes can have the maximal drift difference $2\hat{\rho}$.

Algorithm 2 BP-ISA

procedure *initialize* [in: - / out: -] // N = maximum number of neighbor nodes
  for $\forall i \in \{1,\ldots,N\}$ **do**
    $(T^l_{\mathrm{M}}[i], T^u_{\mathrm{M}}[i]) \leftarrow (-\infty, \infty)$ // most recent bounds from node with index i
    $h_{\mathrm{M}}[i] \leftarrow 0$ // local time when bounds were obtained
  end for
  $n_{\mathrm{M}} \leftarrow 1$ // n_{M} = index of last node encountered

procedure *updateBounds* [in: $(T^l_{\mathrm{old}}, T^u_{\mathrm{old}}), \Delta h$ / out: (T^l, T^u)]
  if $\Delta h \geq 0$ **then**
    $(T^l, T^u) \leftarrow \left(T^l_{\mathrm{old}} + \Delta h/(1+\hat{\rho}), T^u_{\mathrm{old}} + \Delta h/(1-\hat{\rho})\right)$
  else
    $(T^l, T^u) \leftarrow \left(T^l_{\mathrm{old}} + \Delta h/(1-\hat{\rho}), T^u_{\mathrm{old}} + \Delta h/(1+\hat{\rho})\right)$
  end if

procedure *currentBounds* [in: h / out: (T^l, T^u)]
  $(T^l, T^u) \leftarrow \mathit{updateBounds}\left((T^l_{\mathrm{M}}[n_{\mathrm{M}}], T^u_{\mathrm{M}}[n_{\mathrm{M}}]), h - h_{\mathrm{M}}[n_{\mathrm{M}}]\right)$

procedure *intersect* [in: $(T^l_A, T^u_A), (T^l_B, T^u_B)$ / out: (T^l, T^u)]
  $(T^l, T^u) \leftarrow \left(\max(T^l_A, T^l_B), \min(T^u_A, T^u_B)\right)$

procedure *updateMemory* [in: $(T^l, T^u), h$ / out: -]
  for $\forall i \in \{1,\ldots,N\}$ **do**
    $(T^l_{\mathrm{new}}, T^u_{\mathrm{new}}) \leftarrow \mathit{updateBounds}\left((T^l, T^u), h_{\mathrm{M}}[i] - h\right)$
    $(T^l_{\mathrm{M}}[i], T^u_{\mathrm{M}}[i]) \leftarrow \mathit{intersect}\left((T^l_{\mathrm{M}}[i], T^u_{\mathrm{M}}[i]), (T^l_{\mathrm{new}}, T^u_{\mathrm{new}})\right)$
  end for

procedure *generateMessage* [in: h, n / out: $(T^l_c, T^u_c), (T^l_p, T^u_p)$]
  // to node n at local time h
  $\left((T^l_c, T^u_c), (T^l_p, T^u_p)\right) \leftarrow \left(\mathit{currentBounds}(h), (T^l_{\mathrm{M}}[n], T^u_{\mathrm{M}}[n])\right)$

procedure *processMessage* [in: $(T^l_c, T^u_c), (T^l_p, T^u_p), h, n$ / out: -]
  // from node n at local time h
  $\mathit{updateMemory}\left((T^l_p, T^u_p), h_{\mathrm{M}}[n]\right)$
  $(T^l_{\mathrm{M}}[n], T^u_{\mathrm{M}}[n]) \leftarrow \mathit{intersect}\left((T^l_c, T^u_c), \mathit{currentBounds}(h)\right)$
  $h_{\mathrm{M}}[n] \leftarrow h$
  $n_{\mathrm{M}} \leftarrow n$
  $\mathit{updateMemory}(\mathit{currentBounds}(h), h)$

for instance the lower bound is increased at minimal speed (assuming maximal drift $+\hat{\rho}$) to guarantee validity in the future, it is decreased at maximal speed (assuming minimal drift $-\hat{\rho}$) in order to guarantee validity in the past.

The procedure *currentBounds* computes bounds based on the stored bounds for the most recent communication event, which occurred with the remote node with index n_{M}.

The procedure *generateMessage* prepares two sets of bounds: This node's current bounds (T_c^l, T_c^u) and the bounds (T_p^l, T_p^u) for the previous communication event with the remote node identified by the index *n*.

The procedure *processMessage* has additional parameters: T_p^l and T_p^u represent the time bounds the remote node has in memory for the last communication event with this node. The value *n* is used to identify the remote node and to access the stored values $(T_{\mathrm{M}}^l[n], T_{\mathrm{M}}^u[n])$ and $h_{\mathrm{M}}[n]$.

Example

To explain how the BP-ISA works, we use the simple scenario shown in the upper left corner of Figure 13 on page 63:

Two nodes N_1 and N_2 communicate at events *a* and *c*. Node N_1 has an additional communication *b* with a third node that is not shown. At event *a*, nodes N_1 and N_2 exchange their current and previous bounds. Assume that they have never communicated before and therefore the previous bounds are uninitialized $(-\infty, \infty)$. Therefore the first call to *updateMemory* in *processMessage* has no effect on the stored bounds. The nodes combine the received current bounds with their own current bounds and store the result in $(T_{\mathrm{M}}^l[2], T_{\mathrm{M}}^u[2])$ on N_1 and in $(T_{\mathrm{M}}^l[1], T_{\mathrm{M}}^u[1])$ on N_2. The algorithm IM does exactly the same (using $(T_{\mathrm{M}}^l, T_{\mathrm{M}}^u)$).

At event *b*, node N_1 communicates with the third node. We assume that N_1 can improve its current bounds by using the remote node's current bounds. The final call to *updateMemory* thus may improve the stored bounds $(T_{\mathrm{M}}^l[2], T_{\mathrm{M}}^u[2])$ for the communication event *a* with node N_2. Such an improvement constitutes the computation of the first part (from *b* to *a*) of the back path displayed in Figure 13 as a dashed arrow.

At event *c*, N_1 and N_2 exchange their current and previous bounds for event *a*. While the previous bounds of N_2 have not been modified since event *a*, those of N_1 have been improved. When node N_2 thus calls *updateMemory* and then *currentBounds*, the remaining part of the back path is computed.

Computation and memory requirements

The BP-ISA is more complex than the algorithm IM. The computational cost of the BP-ISA per communication event is approximately $(2N+1)$ times the cost of the algorithm IM, where *N* is the number of neighbors a node communicates with. The memory required by the BP-ISA is *N* times that of the algorithm IM.

3.4.3 Performance analysis

We show that the BP-ISA performs at least as well as the algorithm IM and is thus also worst-case-optimal. By means of a simple scenario, we illustrate how and when the BP-ISA can provide better synchronization than the algorithm IM.

Lem. 13: *The BP-ISA always provides equal or better time bounds than the algorithm IM.*

Proof: We remove the calls to *updateMemory* from the procedure *processMessage* of the BP-ISA. Now, $T_{\mathrm{M}}^{l}[n_{\mathrm{M}}]$ takes the role of T_{M}^{l} in the algorithm IM, $T_{\mathrm{M}}^{u}[n_{\mathrm{M}}]$ that of T_{M}^{u} and $h_{\mathrm{M}}[n_{\mathrm{M}}]$ that of h_{M}. It is clear that this modified BP-ISA is equivalent to the algorithm IM.

We now only have to show that the call to *updateMemory* cannot degrade bounds. This procedure modifies the stored bounds only by intersection of the corresponding intervals. It is clear that this never degrades bounds. ■

Cor. 14: *The BP-ISA is worst-case-optimal.*

Improvement over the algorithm IM

The amount of improvement of the BP-ISA is illustrated in Figure 13. It depends on many parameters: the nodes' drifts ρ_1 and ρ_2, the position of event b, and the uncertainty achieved at b in comparison to the uncertainty ΔT at a. These four parameters are explored in the five graphs in Figure 13.

In the upper right corner, event b occurs immediately after a ($x \approx 0$), and it provides zero uncertainty ($y = 0$). In this case, the improvement of the BP-ISA is 100 % if the nodes' drift difference is maximal. The improvement is 0 % if the drift rates are equal.

The graphs in the center row of Figure 13 explore the effect of the position of event b. The average improvement decreases with increasing distance between a and b. The maximal improvement is still 100 % for maximal drift difference.

The graphs in the bottom row of Figure 13 explore the effect of the time uncertainty achieved at b. With increasing uncertainty, the maximal improvement decreases, e.g. at $y = 0.5$, the BP-ISA can achieve 34 % less uncertainty than the algorithm IM if the drift difference is maximal.

We can thus only conclude that depending on the scenario and the drift rates, the improvement of the BP-ISA varies between 0 % and 100 %. In the next section, we examine the *average* improvement for realistic sensor-network scenarios through simulation.

3.4.4 Simulation

In this section, we describe the scenarios that we simulated and the performance measures used to evaluate the synchronization algorithms. We examine the absolute performance of the algorithm IM and of the BP-ISA in these scenarios and quantify the relative improvement of the BP-ISA. The simulations were done using custom C++ programs.

Scenarios and measures

We simulated sensor-network scenarios for a simulation time of 500 h, i.e. almost 21 days. The network contained 100 nodes that were distributed uniformly at random in a square area with edge length 10000. Their local clocks had a constant drift rate between -100 ppm and $+100$ ppm. The number N_A of anchor

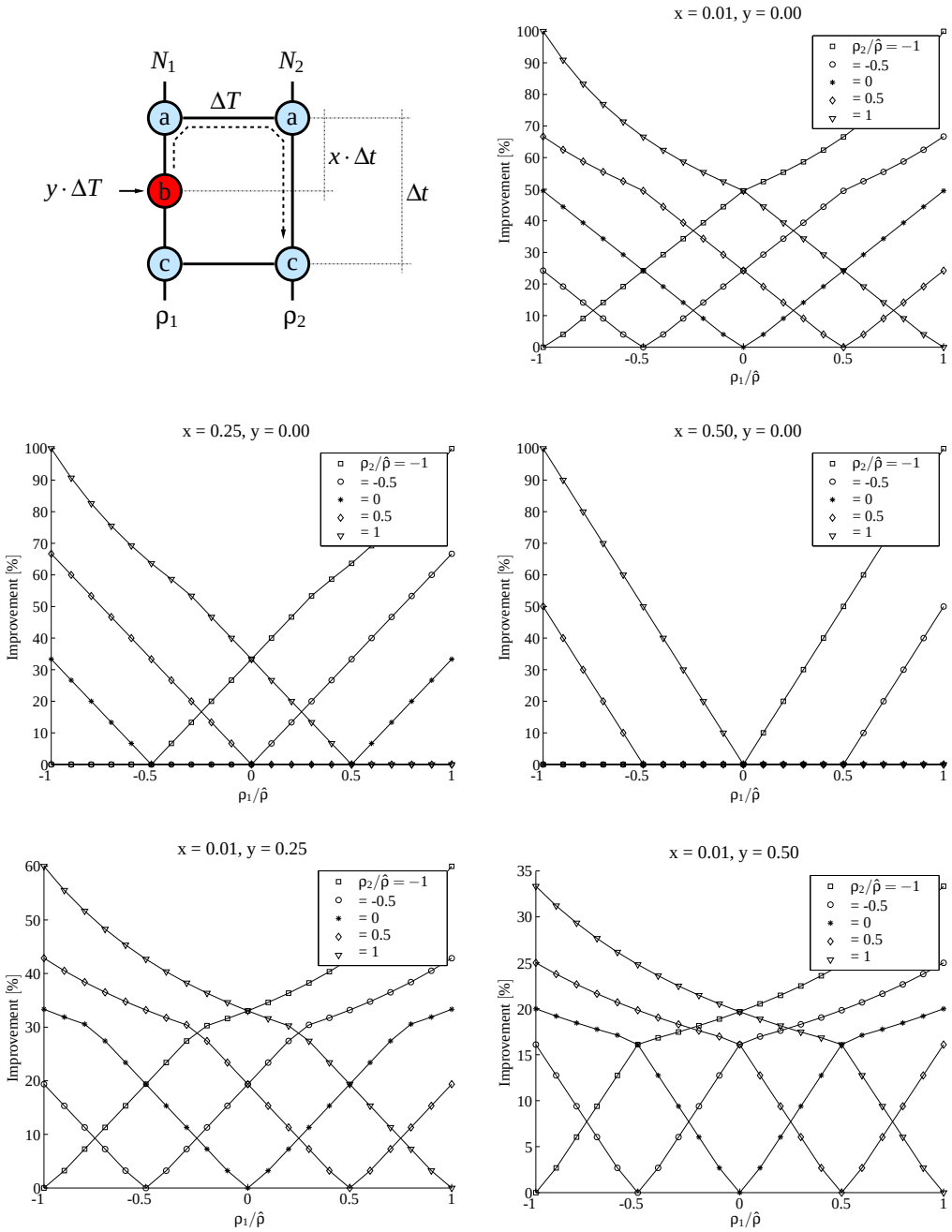

Fig. 13: *Improvement of the BP-ISA in the simple scenario depicted in the upper left corner.* In all figures, the drift rates of both nodes are varied between $-\hat{\rho}$ and $\hat{\rho}$. Each graph shows one combination of the parameters *x* and *y*. The parameter *x* determines the position of event *b*, the parameter *y* determines the time uncertainty achieved in this event. The figure in the upper right corner shows that the BP-ISA can achieve 100 % improvement if event *b* provides uncertainty zero and occurs immediately after event *a*. In the center-row graphs, the position of event *b* is varied, while the bottom-row graphs compare varying uncertainties for the time information provided by this event.

nodes was either $5, 10$ or 20 (out of the total 100 nodes). Two nodes could communicate if they were within each other's transmission range. All nodes had the same transmission range; it was varied between 0.1 and 0.5 times the width of the area in which the nodes were placed. Sensor nodes communicated with other sensor nodes at a variable average frequency of $f_C \in [1/\mathrm{h}, 20/\mathrm{h}]$.[5] Anchor nodes communicated with sensor nodes at a much smaller average frequency $f_A \in [0.002/\mathrm{h}, 2/\mathrm{h}]$.

The performance measure we used was the time uncertainty; we evaluated it after every communication event. Then we took the average over all communication events, but excluded those events after which the nodes had an infinite time uncertainty, i.e. the events from which there existed no path to a source event in the corresponding timing graph. We thus produced the average time uncertainties for the algorithm IM and for the BP-ISA, as shown in Figure 14. From these values, we computed the improvement of the BP-ISA. Figures 15 through 18 give these results as percentages and in milliseconds. For every data point, at least 50 traces were evaluated.

Absolute performance

Figure 14 displays the time uncertainty achieved by the two algorithms. As expected, the uncertainty decreases when communication becomes more frequent. Increasing the communication among sensor nodes decreases the uncertainty to a constant level, while increasing the anchor communication frequency f_A leads to arbitrarily small uncertainties.

Figure 14 shows that for all parameters used in this study, the uncertainty remains in the order of milliseconds. As shown e.g. in [BT02, EGE02, HC02, MFNT00], the delay-induced uncertainty can be reduced to a few microseconds. Therefore our assumption that the delay-induced uncertainty is negligible in comparison to the drift-induced uncertainty is correct in ad-hoc, infrequent-communication networks.

Relative improvement of the BP-ISA

Figures 15 and 16 display the improvement of the BP-ISA for varying transmission ranges. The relative improvement increases significantly up to a ratio between transmission-range and area-width of about 0.15 and then remains roughly constant. Figure 19 explains why: With a small transmission range, sensor nodes are divided into two categories:

1. Nodes that have no communication path to an anchor node. Such nodes have an infinite uncertainty and are not included in the average measure.

2. Nodes that have a communication path to an anchor node and thus have a finite uncertainty. Such nodes are typically very few hops away from an anchor node. Their uncertainty therefore is small and the BP-ISA cannot improve much. Also for large transmission ranges, all sensor nodes are very few hops away from an anchor node.

[5] For $f_C = 10/\mathrm{h}$, 5000 communication events occurred at uniformly random times.

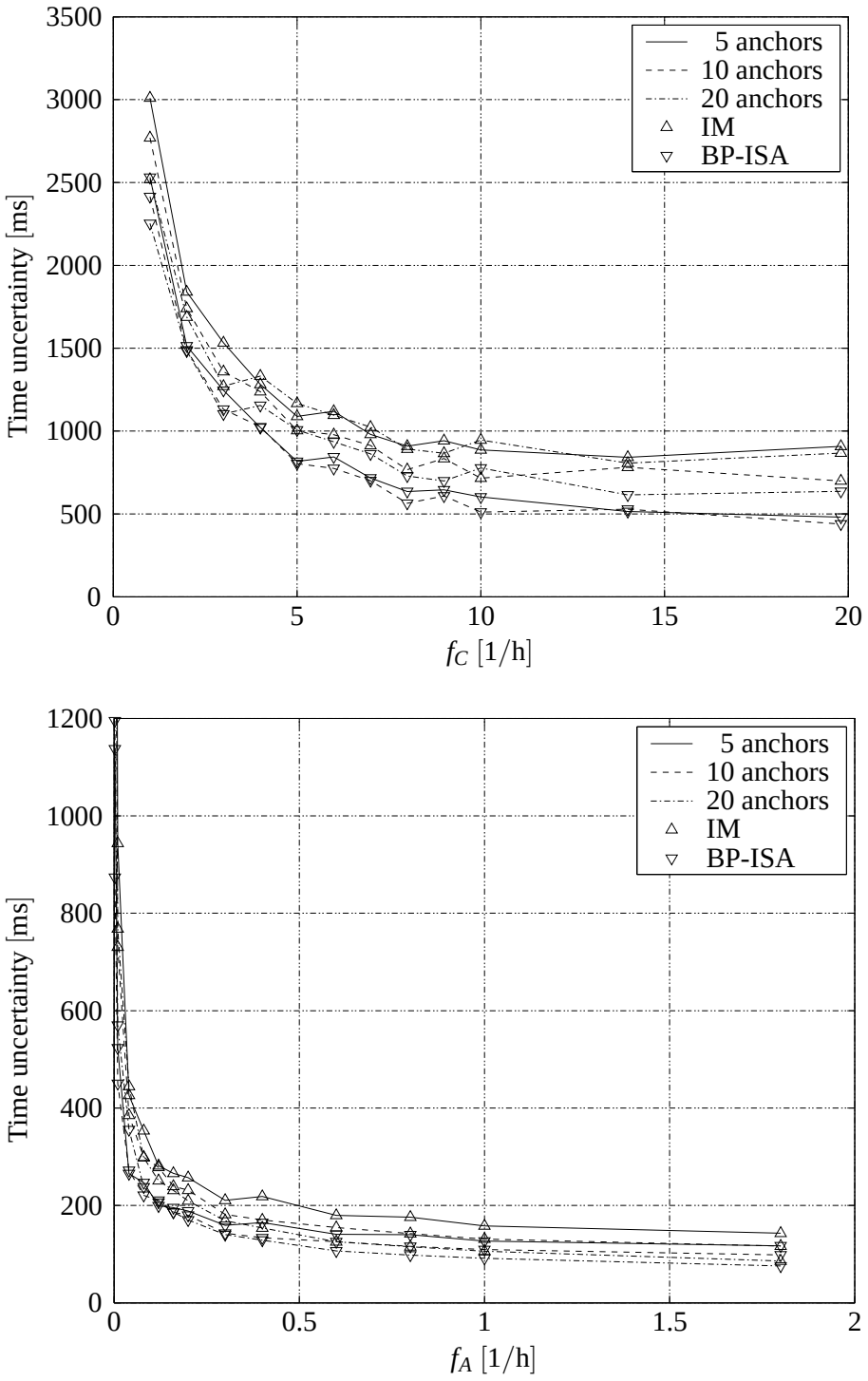

Fig. 14: *Time uncertainty achieved by the algorithm IM and the BP-ISA.* Top: For varying frequencies of communication among sensor nodes. Bottom: For varying frequencies of communication with anchor nodes. The time uncertainty decreases when the sensor nodes communicate more frequently. The effect of the communications with anchor nodes is much stronger.

The largest improvement of the BP-ISA is obtained when many nodes have a path to an anchor, and the average hop distance is large. This is the case for a transmission-range/area-width ratio of about 0.15, which is the range at which almost all nodes are connected.

Figures 17 and 18 display the improvement as a function of sensor and anchor communication frequencies. The improvement of the BP-ISA increases when the sensor nodes communicate more frequently among each other. The improvement decreases if the sensor nodes communicate often with the anchor nodes. It also decreases with increasing number of anchor nodes.

Remarks

The simulation results show that the BP-ISA provides substantial improvements over the algorithm IM, and that these improvements are large in comparison to the delay-induced time uncertainty. The amount of the improvement strongly depends on the scenario: It can be several tens of percents if the sensor nodes communicate relatively often among each other, but only rarely with anchor nodes. The BP-ISA has the greatest advantage over the algorithm IM in networks where the average hop distance to anchor nodes is large. For a given node density, this is the case at those relatively small transmission ranges that lead to just barely connected networks.

Note that the time uncertainty of interval-based algorithms is not directly comparable to the time error of algorithms that compute time estimates. In the average case, the error of an estimate computed as the average of the lower and the upper bound will be much lower than the *guaranteed* uncertainty. Also, the microsecond-range errors reported for instance in [BT02, EGE02, HC02, MFNT00] are achieved in networks with frequent communication.

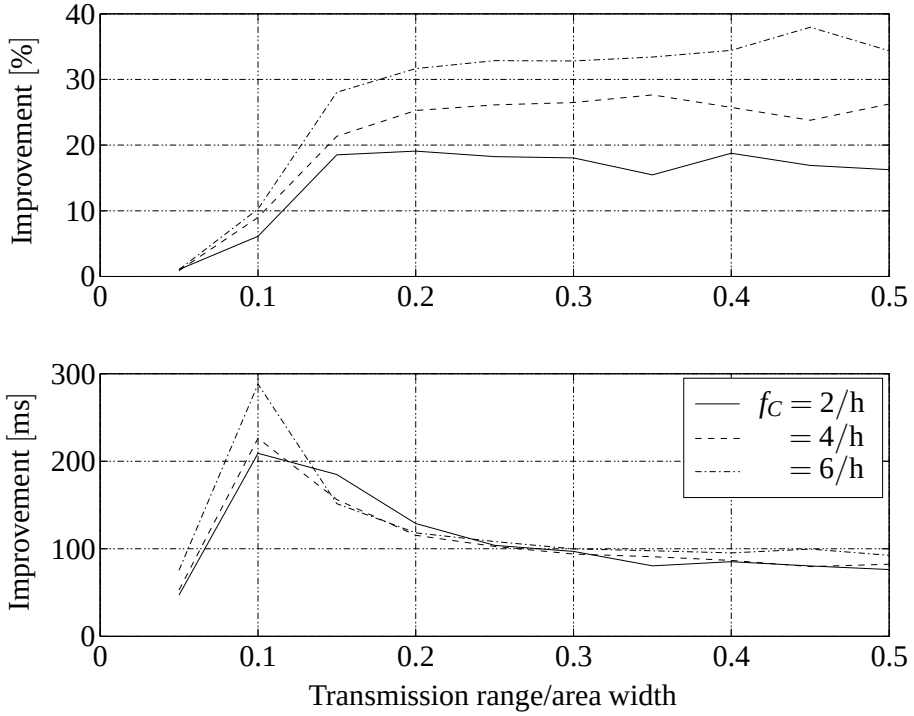

Fig. 15: Improvement of the BP-ISA for varying transmission ranges and communication frequencies f_C. The solid lines in this figure and in Figure 16 represent the same data.

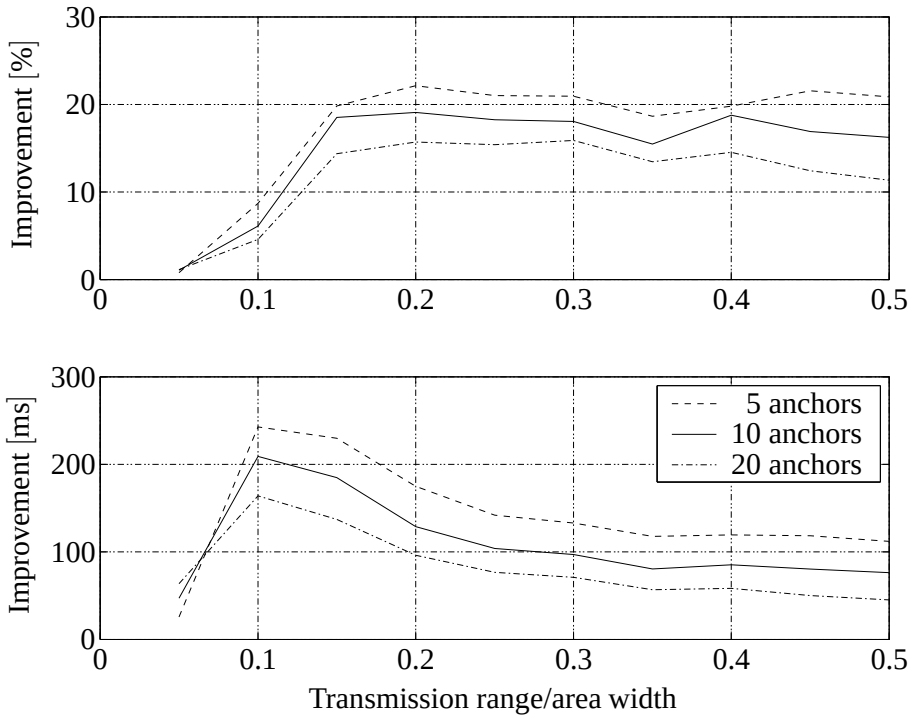

Fig. 16: Improvement of the BP-ISA for varying transmission ranges and numbers of anchor nodes. The solid lines in this figure and in Figure 15 represent the same data.

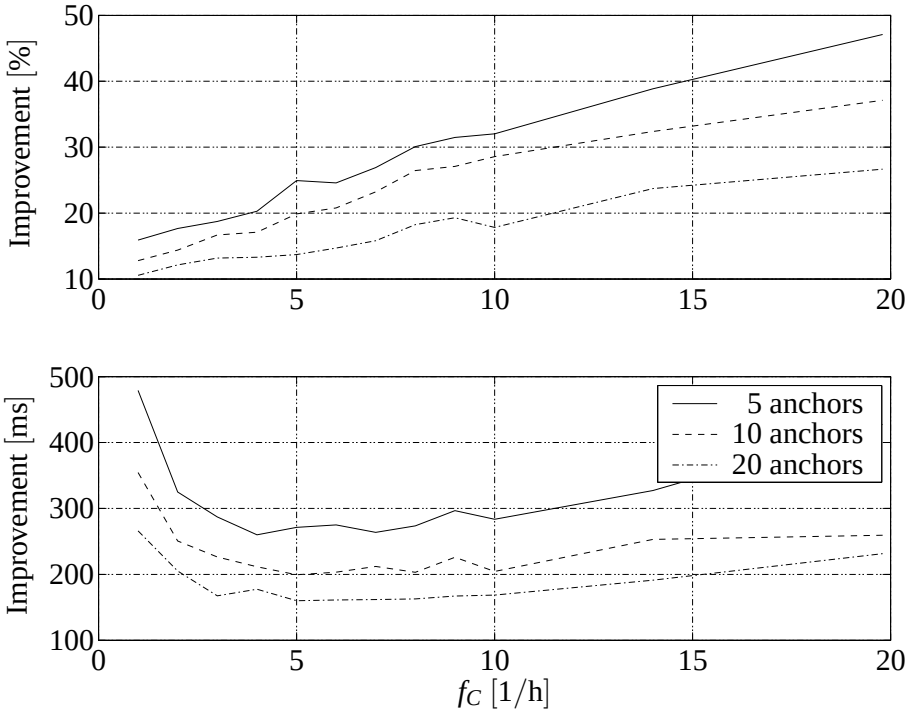

Fig. 17: Improvement of the BP-ISA for varying communication frequencies. With increasing number of communications among the sensor nodes, the improvement grows.

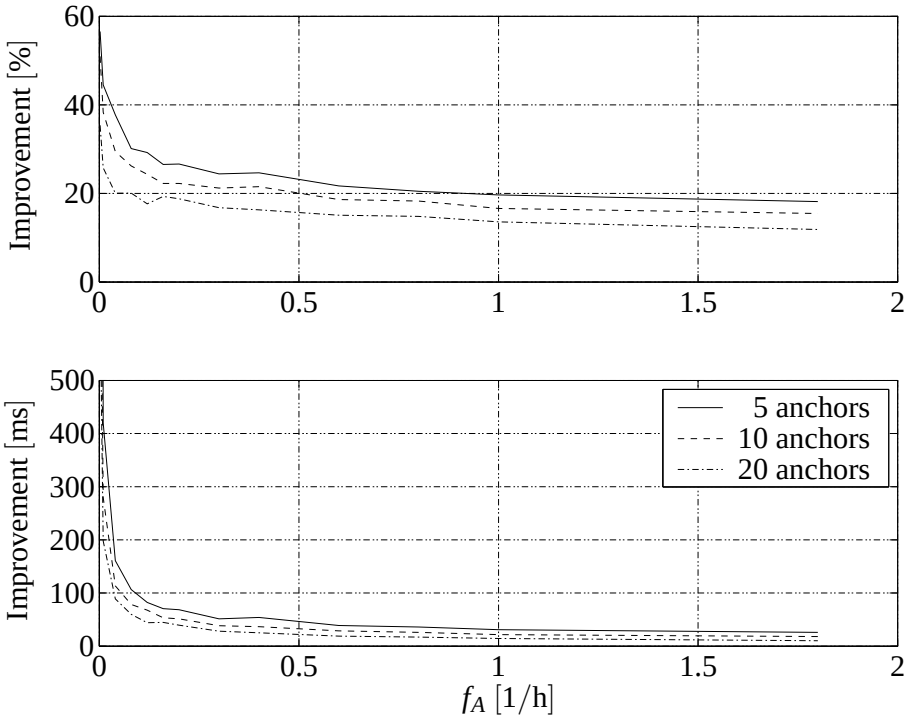

Fig. 18: Improvement of the BP-ISA for varying communication frequencies. With increasing number of communications with anchor nodes, the improvement diminishes.

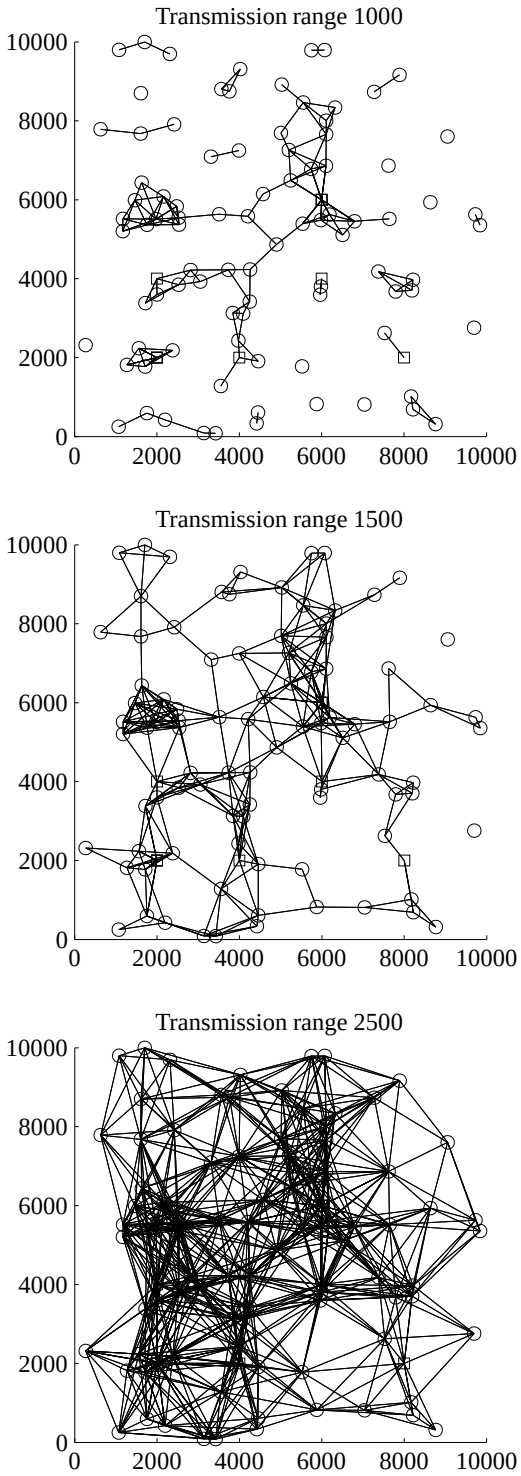

Fig. 19: *Impact of the transmission range.* Anchor nodes are shown as squares, sensor nodes as circles. Top: A small transmission range leads to many partitions in the network. Center: At a transmission range of 0.15 times the width of the area, almost all nodes are connected. As shown in Figures 15 and 16, the (absolute) improvement of the BP-ISA is particularly high for such transmission ranges. Bottom: At higher transmission ranges, most nodes can communicate with an anchor node.

3.5 Internal Synchronization

In this section, we analyze an algorithm for internal synchronization in sensor networks that was proposed in [Röm01]. Our path-based analysis suggests an immediate improvement to the algorithm. While needing less computation and no more communication or memory than the original algorithm, our new algorithm always yields equal or better results and thus outperforms the original algorithm.

We first formally define the system model and state the problem we want to solve. We then revisit the algorithm from [Röm01], propose an improvement, and show that our improved algorithm outperforms the original algorithm. We quantify the improvement and identify the factors its size depends on.

3.5.1 System model

We use the system model from Section 3.2 with some modifications:

Communication model
We adopt the model used in [Röm01] to make our results directly comparable. Namely, we assume a communication event to consist of a single message with a non-zero message delay. Afterwards, we return to the zero-delay model, where a single communication event can contain multiple messages.

Sensor events
A sensor node may receive sensor data from outside the network. We model this as a *sensor event s* which occurs at the sensor node at real time t_s.

Clock drift
For times t_a, t_b with $t_a \neq t_b$, we define the average drift of node N_i in $[t_a, t_b]$ as

$$\bar{\rho}_i(t_a, t_b) = \frac{h^i(t_b) - h^i(t_a)}{t_b - t_a} \quad .$$

Furthermore, each node N_i has its own drift bound $\hat{\rho}_i$.

3.5.2 Problem statement

We now turn to the problem of internal synchronization: Given a trace with an event occurring at node N_j at local time $h^j(t)$, we are interested in tight bounds $H_i^l(t)$, $H_i^u(t)$ on the local time $h^i(t)$ of another node N_i at time t, such that $H_i^l(t) \leq h^i(t) \leq H_i^u(t)$.

In the context of a sensor network, this situation arises, e.g., when a sensor event s is observed at time t_s by node N_j, and another node N_i wants to obtain bounds on $h^i(t_s)$. In the example in Figure 20, the sensor nodes N_2 and N_3 sense a vehicle at local times $h_2(t_1)$ and $h_3(t_2)$.[6] If node N_1 can compute bounds on $h_1(t_1)$ and $h_1(t_2)$ which show that $t_1 < t_2$ (i.e. $H_i^u(t_1) < H_i^l(t_2)$), it can conclude that the vehicle entered the sensing area of N_2 first and then that of N_3.

[6] For simplicity, we assume that a sensor event is only generated at the first sensing of the vehicle, i.e. at the moment where the vehicle enters the sensing area.

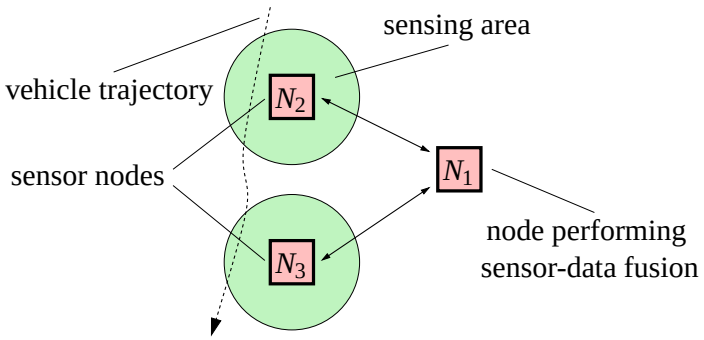

Fig. 20: To perform correct sensor-data fusion, node N_1 has to be able to relate the observations of nodes N_2 and N_3 in time.

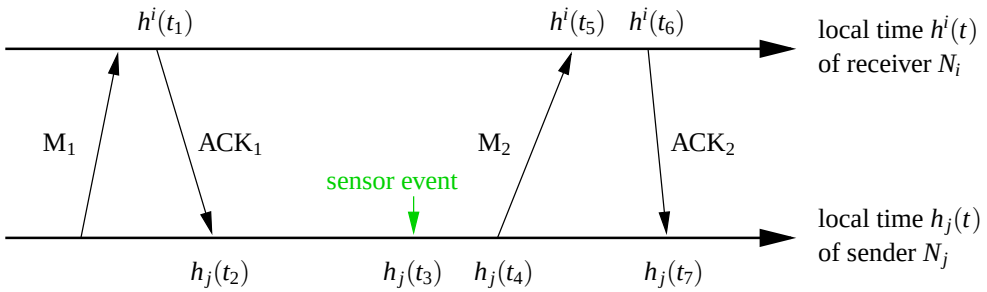

Fig. 21: Message exchange between a sender and a receiver in the algorithm from [Röm01].

3.5.3 The algorithm from [Röm01]

In [Röm01], synchronization is achieved as follows: At some time after the occurrence of a sensor event s at node N_j, this node sends bounds on the local time $h^j(t_s)$ at which the event occurred to node N_i. Node N_i receives these bounds and transforms them into bounds on the local time $h^i(t_s)$ its own clock was showing at time t_s. We will now see in detail how this transformation is done.

The message exchange between a sender and a receiver node is depicted in Figure 21. At time t_3, the sender node N_j observes a sensor event and stores its local time $h^j(t_3)$. At time t_4, node N_j sends $h^j(t_3)$, the local time $h^j(t_2)$ at which the last acknowledgment ACK_1 was received, and the current local time $h^j(t_4)$ to the receiver node N_i in a single message M_2. This message is received by N_i at local time $h^i(t_5)$. Node N_i can now compute bounds on $h^i(t_3)$ by subtracting from $h^i(t_5)$ the maximal and minimal local time that can have elapsed in the real-time interval $[t_3, t_5]$. For the lower bound, this yields

$$H_i^l(t_3) = h^i(t_5) - \left(h^j(t_4) - h^j(t_3)\right)\frac{1+\hat{\rho}_i}{1-\hat{\rho}_j} - \left(\left(h^i(t_5) - h^i(t_1)\right) - \left(h^j(t_4) - h^j(t_2)\right)\frac{1-\hat{\rho}_i}{1+\hat{\rho}_j}\right) \quad . \tag{3.2}$$

The expression on the second line of (3.2) is an upper bound on the local-time difference $h^i(t_5) - h^i(t_4)$, i.e. on the delay of M_2 expressed in local time of N_i. For the upper bound, we have

$$H_i^u(t_3) = h^i(t_5) - \left(h^j(t_4) - h^j(t_3)\right) \frac{1-\hat{\rho}_i}{1+\hat{\rho}_j} \quad . \tag{3.3}$$

3.5.4 Improvement to the algorithm from [Röm01]

Equation (3.2) can be simplified algebraically to

$$H_i^l(t_3) = h^i(t_1) + \left(h^j(t_4) - h^j(t_2)\right) \frac{1-\hat{\rho}_i}{1+\hat{\rho}_j} - \left(h^j(t_4) - h^j(t_3)\right) \frac{1+\hat{\rho}_i}{1-\hat{\rho}_j} \quad .$$

This equation shows that the lower bound is actually computed by first advancing the local time $h^i(t_1)$ by the minimum local time (of node N_i) that passes in the real-time interval $[t_1, t_4]$ (note that $t_2 - t_1 = 0$ is assumed, i.e. the delay of ACK_1 is assumed to be zero), and then subtracting the maximum local time that passes in the interval $[t_3, t_4]$. Informally speaking, we start at t_1, walk past t_2 and t_3 to t_4 and then back to t_3. Our improvement is straightforward: We compute the lower bound $H_i^l(t_3)$ by advancing $h^i(t_1)$ by the minimum local time that passes in the real-time interval $[t_1, t_3]$, i.e.

$$H_i^l(t_3) = h^i(t_1) + \left(h^j(t_3) - h^j(t_2)\right) \frac{1-\hat{\rho}_i}{1+\hat{\rho}_j} \quad . \tag{3.4}$$

We do not change the way the upper bound $H_i^u(t_3)$ is computed, since it is already computed optimally in the original algorithm.

3.5.5 Comparison of the two algorithms

The difference in uncertainty of the two algorithms is

$$\begin{aligned} \Delta U &= \left(h^j(t_4) - h^j(t_3)\right) \left(\frac{1+\hat{\rho}_i}{1-\hat{\rho}_j} - \frac{1-\hat{\rho}_i}{1+\hat{\rho}_j} \right) \\ &= \left(h^j(t_4) - h^j(t_3)\right) \frac{2(\hat{\rho}_i + \hat{\rho}_j)}{1-(\hat{\rho}_j)^2} \quad . \end{aligned}$$

The improvement over the original algorithm grows with $h^j(t_4) - h^j(t_3)$ and with the maximal drifts $\hat{\rho}_i$, $\hat{\rho}_j$. Since our algorithm provides an equal or better uncertainty with less computation and otherwise equal resources, it is strictly superior to the algorithm from [Röm01].

3.6 Optimal Internal Synchronization

Our improved algorithm from Section 3.5.4 does not provide optimal bounds in all cases. In the following, we examine how optimal bounds can always be obtained, albeit at the cost of additional computation, communication, and memory. To this end, we return to the zero-delay model which allows us to combine a message exchange between two nodes into a single, atomic communication event. This is shown in Figure 22, where during each of the communication events a and b, each node sends and receives one message (we do not count acknowledgments as messages here).

In Section 3.6.1, we propose an algorithm which makes use of all the data available for a given communication pattern. It thus provides optimal bounds, as we show in Section 3.6.2.

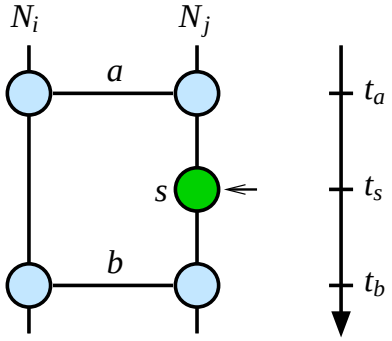

Fig. 22: Two nodes N_i, N_j communicate at events a and b. A sensor event s occurs at node N_j.

3.6.1 An optimal algorithm

Before presenting the algorithm, we identify the general principle by which bounds on local times are computed from bounds on real-time intervals.

General approach

Consider the scenario in Figure 22: To obtain bounds on $h^i(t_s)$, we can compute bounds on the difference $t_s - t_a$ (either directly from $h^j(t_s) - h^j(t_a)$ or using $t_s - t_a = t_b - t_a - (t_b - t_s)$) and then multiply them with $1-\hat{\rho}_i$ and $1+\hat{\rho}_i$, respectively. This results in bounds on $h^i(t_s) - h^i(t_a)$ which we add to $h^i(t_a)$ to obtain bounds on $h^i(t_s)$. We can analogously compute another set of bounds on $h^i(t_s)$ from $t_b - t_s$, and finally choose the best bounds.

In a more complex scenario, such as the one in Figure 23, there may be various bounds Δ^l, Δ^u on the time differences that we use to compute bounds $H_i^l(t_s)$, $H_i^u(t_s)$ on the local time $h^i(t_s)$. The bounds Δ^l, Δ^u can be illustrated by paths in the event chart as shown in Figure 23.

Paths in the event chart

In the scenario depicted in Figure 23, we can compute bounds on $h^i(t_s)$ simply by using the short path s–b. However, we then do not use any information from the events c and d. In the ideal case of maximum drift diversity between nodes

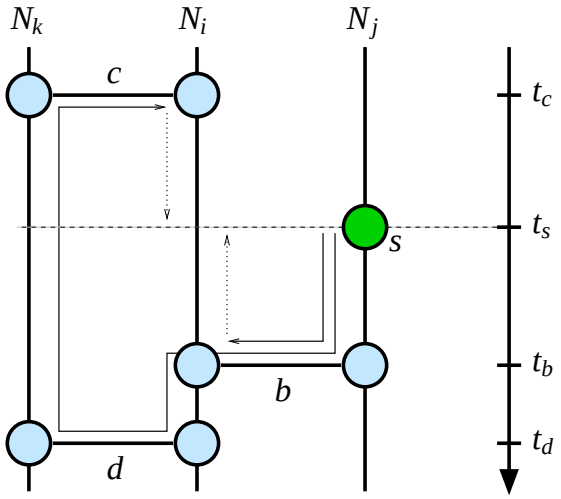

Fig. 23: Three nodes N_i, N_j, N_k with communication events $b,\ldots,d$. A sensor event s occurs at node N_j. The communication events c and d may help to find better bounds on $h^i(t_s)$. The dotted lines indicate the real-time intervals on which bounds are computed using the paths shown by the corresponding solid lines.

N_i and N_k (i.e. $\rho_i = \pm\hat{\rho}_i$ and $\rho_k = \mp\hat{\rho}_k$), the uncertainty about $h^i(t_s)$ can be reduced with this additional information, because $\overline{\rho}_i(t_c,t_d)$ is known, and hence also $\overline{\rho}_i(t_s,t_b)$. This knowledge is implicitly taken into account if we compute for instance the lower bound $H_i^l(t_s)$ as $h^i(t_c)+\Delta^l[c,s](1-\hat{\rho}_i)$, where we derive $\Delta^l[c,s]$ along the long path *s*–*b*–*d*–*c*.

Optimal algorithm

The principle of the optimal algorithm consists in computing the bounds on $h^i(t_s)$ using all possible paths in the event chart which start at event s on node N_j and arrive at some event on node N_i. This approach is elegant in that all improvements due to drift diversity are "automatically" taken into account by traversing all possible paths and choosing the best, i.e. the one that provides the best bound. The quality of a path or of a path segment depends on how close the assumed drift on it is to the actual drift. If the two are equal, we call the path or path segment *tight*. If a bound on $h^i(t_s)$ results from a tight path, then the bound is clearly equal to $h^i(t_s)$.

The optimal algorithm is given below as Algorithm 3. It obviously requires a lot of communication, memory, and computation. We present it as the theoretical reference for the maximum synchronization quality that can be achieved. We now show the algorithm's optimality.

3.6.2 Optimality of Algorithm 3

We now show that in our model, no correct, deterministic algorithm can obtain better bounds than those given by Algorithm 3.

Algorithm 3 Optimal internal synchronization by evaluation of all paths

Let a communication scenario between n network nodes $N_1, \ldots, N_n$ be given. Assume that during communication events, the two nodes involved exchange their complete views. Then for any event s that occurs at time t_s at node N_j and not at node N_i, node N_i can compute tight bounds $H_i^l(t_s)$, $H_i^u(t_s)$ on the local time $h^i(t_s)$ from its maximal (i.e. latest) view as follows:

1. For any event e at node N_i, N_i computes bounds $\Delta_i^l[e,s]$, $\Delta_i^u[e,s]$ on $t_s - t_e$ by traversing all paths between s and e and choosing the best among all the resulting bounds. We illustrate this in Figure 24 for $\Delta^l[c,s]$ and $\Delta^u[d,s]$.

2. Bounds on $h^i(t_s)$ are computed from the bounds obtained in Step 1: For every event e, we obtain bounds according to (we use $[x]^+$ as an abbreviation for $\max\{x,0\}$)

$$H_i^l(t_s) \geq h^i(t_e) + \left[\Delta_i^l[e,s] \cdot (1-\hat{\rho}_i)\right]^+ - \left[-\Delta_i^l[e,s] \cdot (1+\hat{\rho}_i)\right]^+$$
$$H_i^u(t_s) \leq h^i(t_e) + [\Delta_i^u[e,s] \cdot (1+\hat{\rho}_i)]^+ - [-\Delta_i^u[e,s] \cdot (1-\hat{\rho}_i)]^+ \quad .$$

   In each of the two expressions above, only one of the maximization operations can yield a value strictly greater than zero (if we make the reasonable assumption that $\hat{\rho}_i < 1$), depending on whether e precedes or succeeds s.

3. The final bounds $H_i^l(t_s)$, $H_i^u(t_s)$ on $h^i(t_s)$ are given as the maximum of all lower and the minimum of all upper bounds obtained in Step 2.

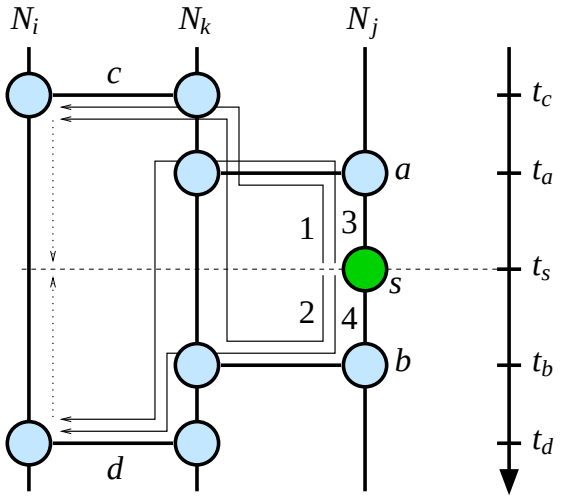

Fig. 24: Illustration of paths in the event chart. Four different paths for computing a lower bound $H_i^l(t_s)$ on $h^i(t_s)$ are shown. The corresponding formulas are given in Figure 25. Since a cycle in a path cannot improve the bound computed from it, the optimal path for the lower bound has to be one of the paths shown; the same holds for the upper bound.

$$1: \quad \Delta^l[c,s] \geq \frac{h^j(t_s) - h^j(t_a)}{1+\hat{\rho}_j} + \frac{h_k(t_a) - h_k(t_c)}{1+\hat{\rho}_k}$$

$$2: \quad \Delta^l[c,s] \geq -\frac{h^j(t_b) - h^j(t_s)}{1-\hat{\rho}_j} + \frac{h_k(t_b) - h_k(t_c)}{1+\hat{\rho}_k}$$

$$3: \quad \Delta^u[d,s] \leq \frac{h^j(t_s) - h^j(t_a)}{1+\hat{\rho}_j} - \frac{h_k(t_d) - h_k(t_a)}{1-\hat{\rho}_k}$$

$$4: \quad \Delta^u[d,s] \leq -\frac{h^j(t_b) - h^j(t_s)}{1+\hat{\rho}_j} - \frac{h_k(t_d) - h_k(t_b)}{1+\hat{\rho}_k}$$

Fig. 25: Formulas for the bounds in Figure 24.

Thm. 15: **(Optimality of Algorithm 3)** *Let a trace T, a sensor event s occurring at node N_j, and another node N_i be given. Then no correct deterministic algorithm exists that provides better bounds on $h^i(t_s)$ than Algorithm 3.*

Proof: We prove Theorem 15 by contradiction: Assume that given the maximal (i.e. latest) views of all nodes resulting from trace T, subsumed in a total view V, an algorithm A provides the bounds $\hat{H}_i^l(t_s)$ and $\hat{H}_i^u(t_s)$ which are correct in T, i.e. $\hat{H}_i^l(t_s) \leq h^i(t_s) \leq \hat{H}_i^u(t_s)$. Further assume that algorithm A provides better bounds than Algorithm 3, i.e. either $\hat{H}_i^l(t_s) > H_i^l(t_s)$ or $\hat{H}_i^u(t_s) < H_i^u(t_s)$, where $H_i^l(t_s), H_i^u(t_s)$ are the bounds computed by Algorithm 3.

As we will show in Lemma 16 below, it is always possible to construct an admissible trace T' with view V (i.e. T' is indistinguishable from T) in which $h^i(t_s) = H_i^l(t_s)$. Thus, Algorithm 3 provides a tight lower bound. Equally, it

is always possible to construct an admissible trace T'' with view V in which $h^i(t_s) = H_i^u(t_s)$. Thus, Algorithm 3 provides a tight upper bound.

Therefore, algorithm A is not correct for either the trace T' (since $h^i(t_s) = H_i^l(t_s) < \hat{H}_i^l(t_s)$) or the trace T'' (since $h^i(t_s) = H_i^u(t_s) > \hat{H}_i^u(t_s)$). This contradicts our initial assumption. ■

Construction of indistinguishable traces

In the proof of Theorem 15, we required that for a given trace T, it is possible to construct an admissible and indistinguishable trace T' in which $h^i(t_s) = H_i^l(t_s)$. The trace T' is constructed by making the path leading to the lower bound $H_i^l(t_s)$ tight; this is achieved by adjusting the drift rates along this path and shifting the real times of the events limiting the path segments. For an event e occurring at time t_e in trace T, we write t'_e for the time at which e occurs in T'. Shifting the real time of a communication event always affects two nodes and might conceivably lead to a violation of (2.1). We will now show that this cannot happen.

Lem. 16: *Let an admissible trace T with view V and a sensor event s occurring at time t_s at node N_j be given. For any event e occurring at node N_i, let $\Delta^l[e,s]$ be the lower bound on $t_s - t_e$ computed by Algorithm 3. Then an admissible trace T' with view V' can be constructed such that $V' = V$ and the lower bound $\Delta^l[e,s]$ is equal to $t'_s - t'_e$ in T'.*

Proof: If $\Delta^l[e,s] = t_s - t_e$ in trace T, we set $T' = T$ and are done. If $\Delta^l[e,s] < t_s - t_e$, let p_{best} be the best path which is used by Algorithm 3 to compute $\Delta^l[e,s]$. If there is no other path from s to e, then all path segments along p_{best} can be made tight without influencing other paths, and we are done.

If there is at least one other path, we have a cycle and another path might be influenced by the tightening of p_{best}. Let event j (for join) be an event on the best path p_{best} at which another path p_{other} joins. Let event f (for fork) be the latest event on p_{best} before j which is also on p_{other}. In Figure 27, we might for instance have $e = j = a$, $f = s$, $p_{\text{best}} = s$–a, and $p_{\text{other}} = s$–d–c–a.

Let Δ^l_{best} and Δ^l_{other} be the lower bounds defined by p_{best} and p_{other}, respectively. Since p_{best} is the best path, we have

$$\Delta^l_{\text{best}}[j,f] \geq \Delta^l_{\text{other}}[j,f] \ . \tag{3.5}$$

In the trace T', the best path shall be tight, i.e. $\Delta^l_{\text{best}}[j,f] = \Delta[j,f] = t'_f - t'_j$. We set the times at which events f and j occur in T' to $t'_f = t_f$ and

$$t'_j = t_f - \Delta^l_{\text{best}}[j,f] \geq t_f - (t_f - t_j) = t_j \ . \tag{3.6}$$

To show that T' is admissible, we have to show that $\Delta^l_{\text{other}}[j,f]$ is a valid lower bound on $t_f - t'_j$ in T', even though $t'_j \geq t_j$. This follows from (3.5) and (3.6): $t_f - t'_j = \Delta^l_{\text{best}}[j,f] \geq \Delta^l_{\text{other}}[j,f]$. ■

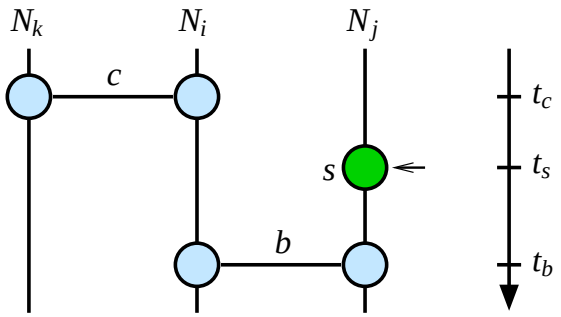

Fig. 26: There are no cycles in the event chart. It is therefore straightforward that all path segments can be made tight.

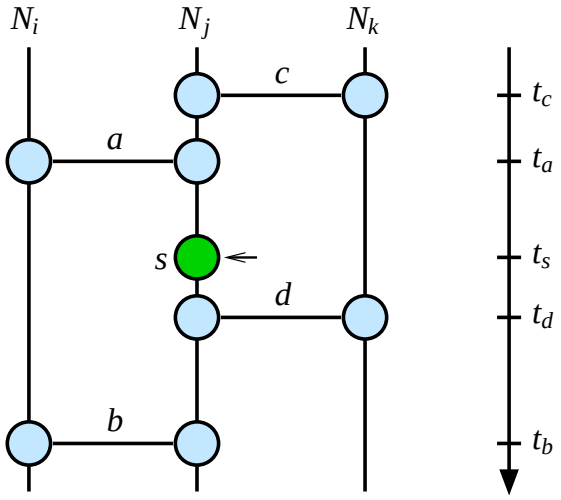

Fig. 27: Several paths form cycles; therefore it is not a priori clear whether clock drifts and event times can always be modified such that $H_i^l(t_s) = h^i(t_s)$.

Cor. 17: *We can achieve $H_i^l(t_s) = h_i(t_s)$ in trace T'. According to Lemma 16, the best path from s to some event e on node N_i can be made tight. Analogously, we can also set $\rho_i = \pm\hat{\rho}_i$ between events s and e, thus achieving $H_i^l(t_s) = h_i(t_s)$.*

We will now illustrate Theorem 15 with two examples. In Figure 26, there is only one path; no matter how large and complex the portion of the event chart to the left of event c at node N_i is, we can shift t_c arbitrarily as long as there is only one path from s to N_i. This also applies to b and t_b here. Thus, $H_i^l(t_s) = h^i(t_s)$ can always be achieved.

In Figure 27, there is more than one path from s to N_i. The best path used by Algorithm 3 forms at least one cycle with some other path. Thus, the real times of the events along the best path cannot be shifted arbitrarily. In the following, we argue that they can always be shifted enough to make the best path tight. Let us assume that the lower bound on $h^i(t_s)$ provided by Algorithm 3 is $H_i^l(t_s) = h^i(t_a) + \Delta^l[a,s](1-\hat{\rho}_i)$, and that the path s–a from which $\Delta^l[a,s]$ is derived is

not tight, i.e. $\Delta^l[a,s] < t_s - t_a$. To obtain the trace T', we therefore increase t_a until $\Delta^l[a,s] = t_s - t_a$. To keep $h^j(t_a) - h^j(t_c)$ constant, we need to decrease $\overline{\rho}_j(t_c,t_a)$, increase t_c, or both. Suppose that $\overline{\rho}_j(t_c,t_a) = -\hat{\rho}_j$ and hence cannot be decreased. We hence have to increase t_c. To keep $h_k(t_d) - h_k(t_c)$ constant, we need to increase $\overline{\rho}_k(t_c,t_d)$, increase t_d, or both. Suppose that $\overline{\rho}_k(t_c,t_d) = +\hat{\rho}_k$ and hence cannot be increased. We hence have to increase t_d. To keep $h^j(t_d) - h^j(t_s)$ constant, we need to decrease $\overline{\rho}_j(t_s,t_d)$ (we cannot modify t_s). Suppose that $\overline{\rho}_j(t_s,t_d) = -\hat{\rho}_j$ and hence cannot be decreased. Now, it would seem that we cannot construct T' with $H_i^l(t_s) = h^i(t_s)$. Actually, we already have T': We assumed

$$\overline{\rho}_j(t_c,t_a) = -\hat{\rho}_j \qquad \overline{\rho}_k(t_c,t_d) = +\hat{\rho}_k \qquad \overline{\rho}_j(t_s,t_d) = -\hat{\rho}_j \ .$$

These assumptions provide us with a tight path *s*–*d*–*c*–*a* and thus a tight bound

$$\begin{aligned}\Delta^l[a,s] &= -\frac{h^j(t_d) - h^j(t_s)}{1-\hat{\rho}_j} + \frac{h^j(t_d) - h^j(t_c)}{1+\hat{\rho}_j} - \frac{h^j(t_a) - h^j(t_c)}{1-\hat{\rho}_j} \\ &= t_s - t_a \ .\end{aligned}$$

Obviously, the path *s*–*a* cannot be made tight if there already is another tight path *s*–*d*–*c*–*a*. Our initial assumption that *s*–*a* is the best path used by Algorithm 3 was therefore wrong.

3.7 Optimal and Efficient External Synchronization

In this section, we will first present an optimal algorithm for external synchronization. The algorithm operates analogously to the one for internal synchronization presented in Section 3.6: Complete views are stored, communicated, and used for the computation of best bounds. We then quantify the complexity of the algorithm by presenting simulation results for its running time. Finally, we investigate how limiting the size of the views affects running time and synchronization quality.

3.7.1 The optimal algorithm

The variant of Algorithm 3 for external synchronization is given below as Algorithm 4. As in the algorithm IM and the BP-ISA, bounds on the real time t_d at which a destination event *d* occurs are obtained by using all source events in the view. As the similarity between Algorithms 3 and 4 illustrates, external and internal synchronization both share the computation of bounds on the length of real-time intervals. In external synchronization, these bounds are used to appropriately advance bounds on the system time, while in internal synchronization bounds on other nodes' local times are advanced.

Algorithm 4 Optimal external synchronization by evaluation of all paths

Let a communication scenario between n network nodes $N_1, \ldots, N_n$ be given. Assume that during communication events, the two nodes involved exchange their complete views. Then for any event d that occurs at time t_d at node N_i, node N_i can compute tight bounds $T^l(t_d)$, $T^u(t_d)$ on the real time t_d from its maximal (i.e. latest) view as follows:

1. For any source event s in node N_i's view, N_i computes bounds $\Delta_i^l[s,d]$, $\Delta_i^u[s,d]$ on $t_d - t_s$ by traversing all paths between s and d and choosing the best among all the resulting bounds.

2. Bounds on t_d are now computed according to (we use $[x]^+$ as an abbreviation for $\max\{x,0\}$)

$$T^l(t_d) = T^l(t_s) + \left[\Delta_i^l[s,d]\right]^+ - \left[-\Delta_i^l[s,d]\right]^+$$
$$T^u(t_d) = T^u(t_s) + [\Delta_i^u[s,d]]^+ - [-\Delta_i^u[s,d]]^+ \quad .$$

   In each of the two expressions above, only one of the maximization operations can yield a value strictly greater than zero, depending on whether s precedes or succeeds d.

We simulated Algorithm 4 with parameters as in Section 3.4.4 to make the results comparable: during a simulation time of 500 h, 100 nodes distributed uniformly at random in a square area with edge length 10000 communicate with each other. We fixed the number of anchor nodes (out of the total 100 nodes) at 10, and the transmission range at 1500. As we discussed in Section 3.4.4, this is the range where all nodes are just barely connected and the advantage of the BP-ISA over the algorithm IM is the largest. The anchor nodes communicated with non-anchor nodes at an average frequency f_A of $0.02/\text{h}$.

The running time of the simulation for a varying number of communication events between non-anchor nodes is shown in Figures 28 and 29. Both figures represent the same data; the logarithmic y-axis in Figure 29 exposes the high variability of the running time for small numbers of communication events. This variability is due to the fact that the communication events were generated at random, and for a small number of events the connectivity and hence the structure of the event chart greatly depends on the choice of events.

We assume the running time of the simulation to be proportional to the average running time of the algorithm on the individual nodes. Note that we do not modify the number of nodes, but only the number of communication events and hence the number of paths used in the computation. Hence, the proportionality factor between the running times of the total simulation and of the algorithm on an individual node is constant.

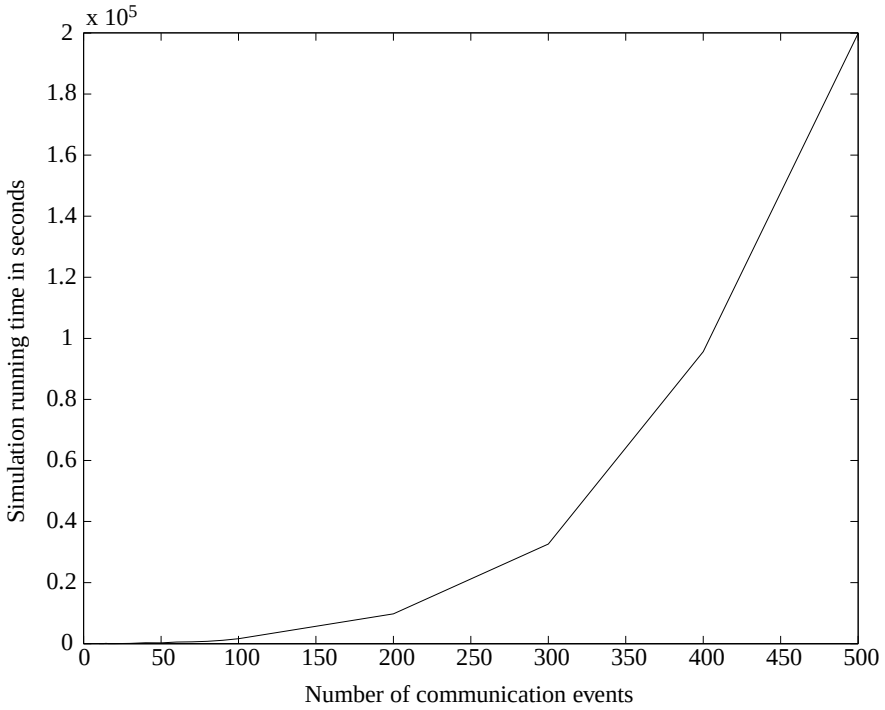

Fig. 28: Total running time of the simulation of the optimal algorithm in seconds for a varying number of communication events between non-anchor nodes.

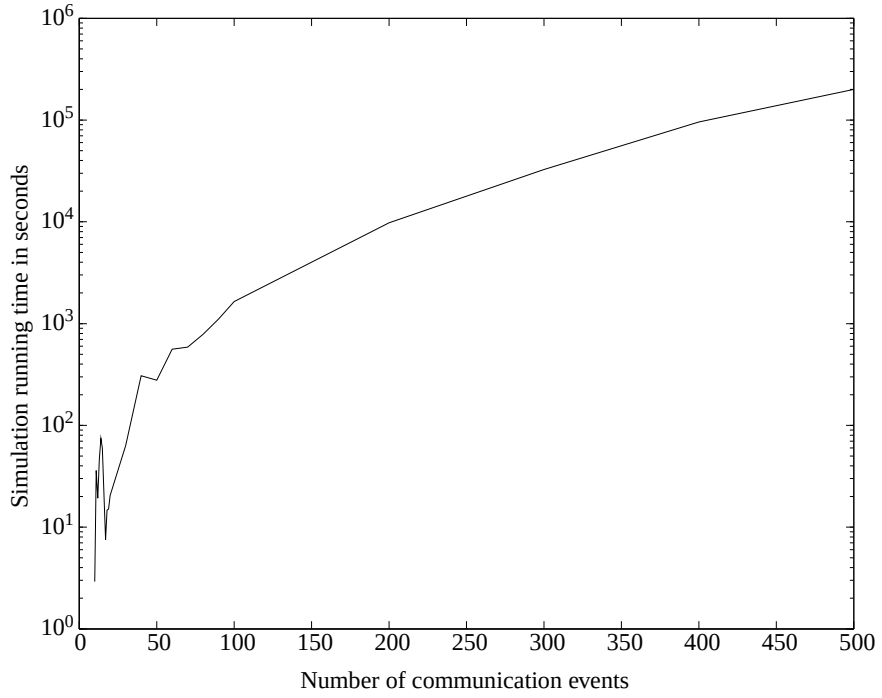

Fig. 29: The data are the same as in Figure 28; the logarithmic y-axis exposes the high variability of the running time for small numbers of communication events.

3.7.2 Running time with limited view size

The running time of the optimal algorithm is prohibitive in all but the most primitive scenarios; we therefore investigate the effect of limiting the size of nodes' views and thus the number of paths that are used by the algorithm. Both the running time and the synchronization quality should decrease with decreasing view size; there is a trade-off between computation and storage capacity used and the achievable synchronization quality.

We now present simulation results for the running time of the algorithm with limited view size. More precisely, each node stores only a limited number of events per node. As an example, assume that in a network with 100 nodes the view size is limited to 10 events per node. This means that any node will store at most 10 of its own events and 10 events for each other node in the network, i.e. at most 1000 events. Figures 30 through 35 show the simulation running time as a function of the view size for different numbers of communication events.

Figure 30 shows that for 10 communication events per node, storing more than 20 events per node does not lead to a further increase in running time. With 10 communications initiated by each node, on average a node participates in 20 communications, and there are only 20 events per node to be stored.

For 20 communication events (Figure 31), the curve flattens at 40 stored events per node, as we can conclude by observing that the running time for 40 stored events equals the one for unlimited view size.

Figures 32 through 35 illustrate how after an initial fast growth the running time increases linearly with the view size. This is exactly the behavior of the leftmost portions of the curves in Figures 30 and 31 ($x \in [0, 10]$).

For 1000 communication events per node, the simulation with unlimited view size could not be performed.

3.7.3 Time uncertainty with limited view size

We now present results for the time uncertainty achieved with limited view size. Figure 36 shows the percentage by which the uncertainty achieved by the optimal algorithm is smaller than that achieved by the algorithm IM. The ideal view size (beyond which additional stored events do not further reduce the uncertainty) increases with increasing number of communications per node. In a scenario with very little communication (50 events per node), the improvement is comparatively small and reaches its maximal value already at a view size of approximately 6 events per node. When communication is more frequent (1000 events per node), we observe a relatively fast increase in the improvement up to a view size of approximately 13 events per node.

The reason why there is a finite ideal view size is that the time information gathered from old events is superceded by time information from newer events.

Before the improvement reaches its maximal value, there is a trade-off between computation and memory used and the achievable synchronization quality. As the view size increases, the running time of the algorithm increases considerably, while the improvement in synchronization quality is moderate.

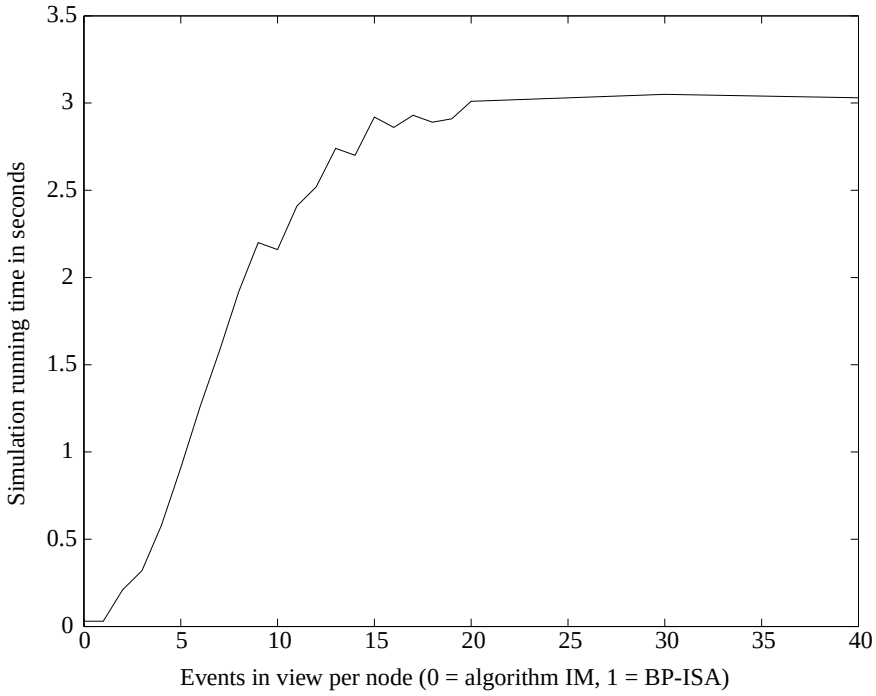

Fig. 30: Total running time of the simulation of the optimal algorithm in seconds for a varying number of events per node in the view. The number of communication events per node was 10, the running time with unlimited view size was 2.92 s.

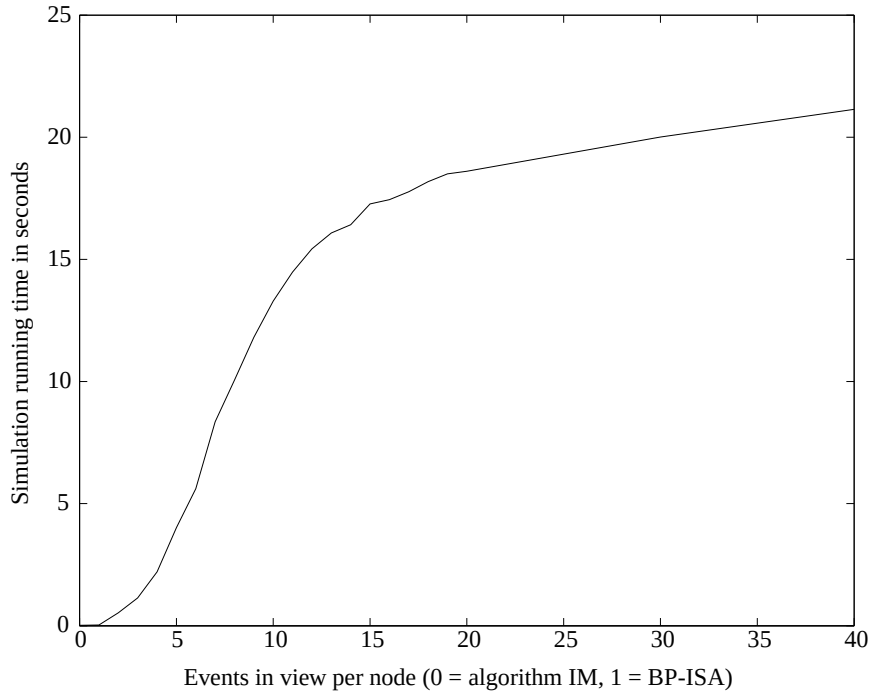

Fig. 31: Total running time of the simulation of the optimal algorithm in seconds for a varying number of events per node in the view. The number of communication events per node was 20, the running time with unlimited view size was 20.53 s.

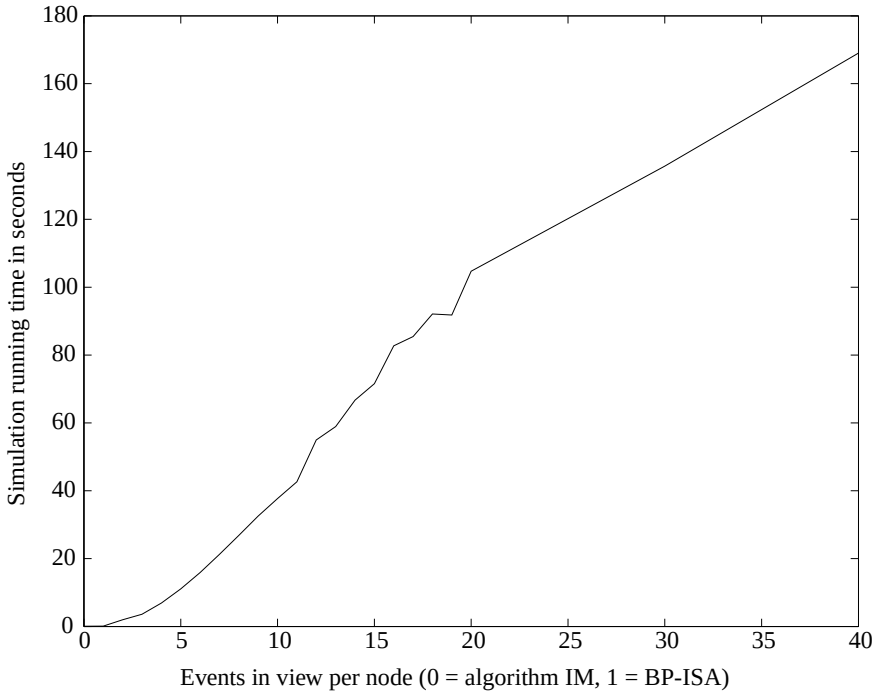

Fig. 32: Total running time of the simulation of the optimal algorithm in seconds for a varying number of events per node in the view. The number of communication events per node was 50, the running time with unlimited view size was 277.82 s.

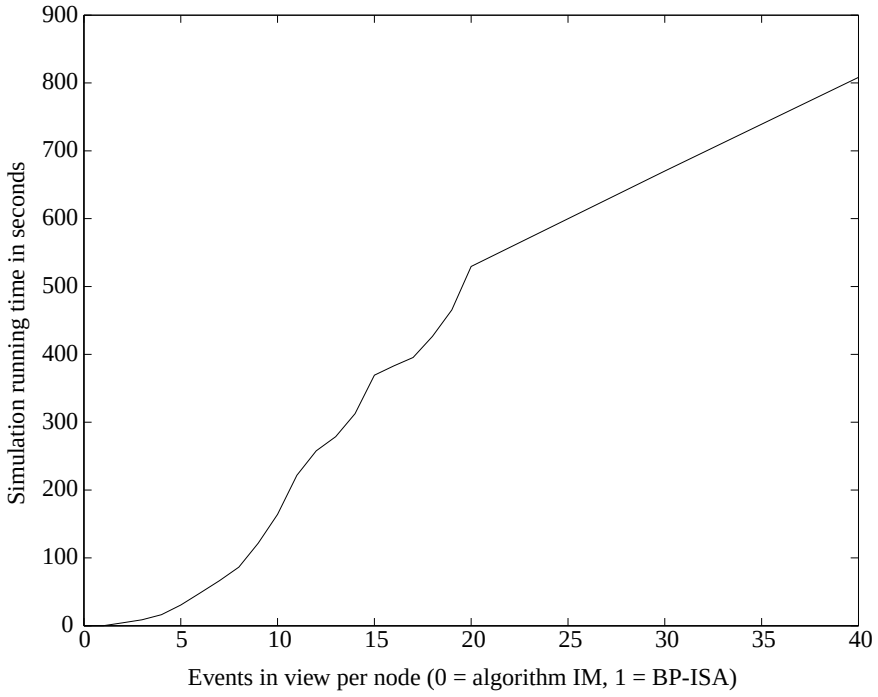

Fig. 33: Total running time of the simulation of the optimal algorithm in seconds for a varying number of events per node in the view. The number of communication events per node was 100, the running time with unlimited view size was 1648.37 s.

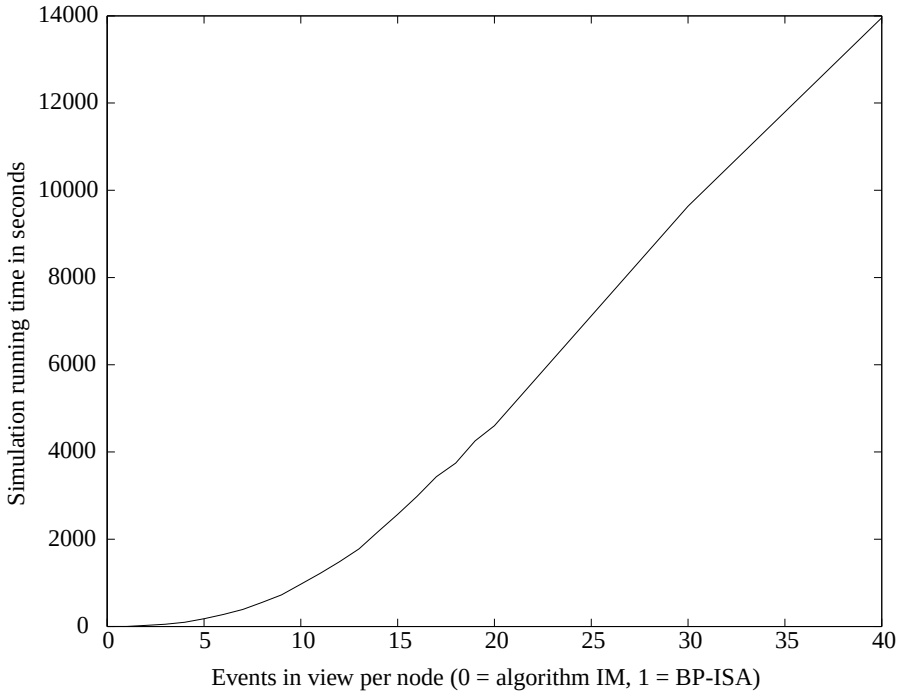

Fig. 34: Total running time of the simulation of the optimal algorithm in seconds for a varying number of events per node in the view. The number of communication events per node was 500, the running time with unlimited view size was 199897.60 s.

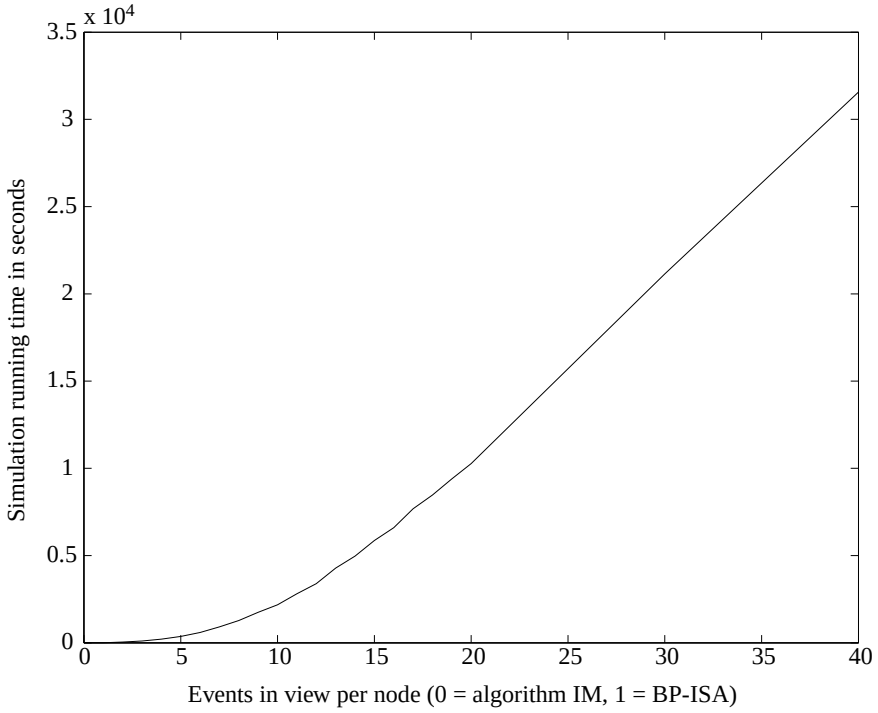

Fig. 35: Total running time of the simulation of the optimal algorithm in seconds for a varying number of events per node in the view. The number of communication events per node was 1000, the simulation with unlimited view size was not performed.

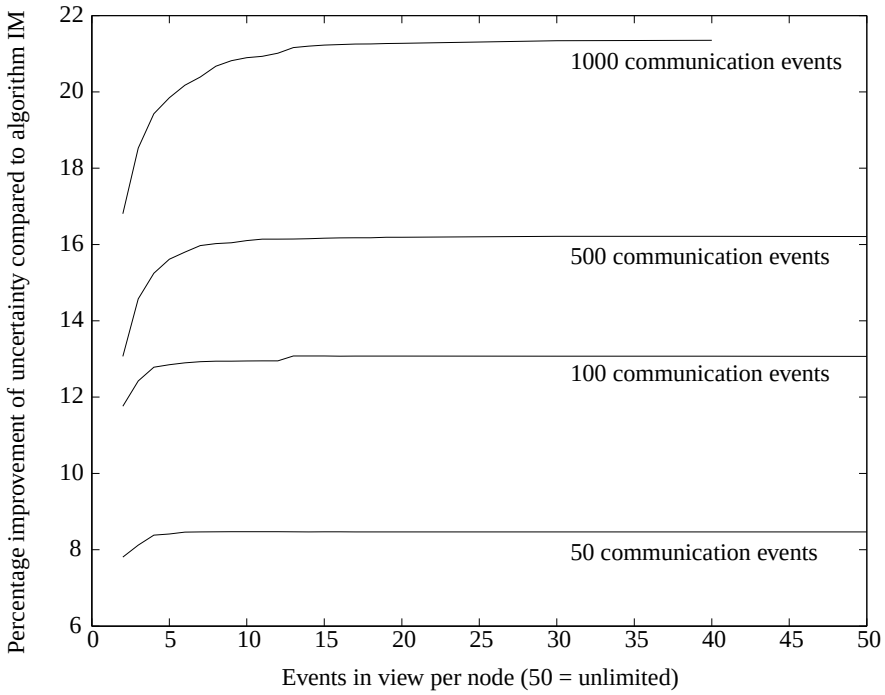

Fig. 36: Percentage improvement of the optimal algorithm for a varying number of events per node in the view. The number of communication events per node was 50, 100, 500, and 1000. For 1000 events, the simulation with unlimited view size was not performed.

3.8 Node Mobility

In the following, we show that node mobility puts no burden on interval-based synchronization, but actually helps it. Examples of mobility comprise nodes attached to animals or humans or nodes transported by wind or water from one location to another. Recent work has addressed the special characteristics of synchronization in mobile ad-hoc networks [Röm01, SY04]. To our knowledge, the impact of mobility on synchronization has not yet been considered.

Many synchronization algorithms tailored to mobile ad-hoc networks have been proposed recently; most of them use time estimates. Since it is not a priori clear how two time estimates should be combined, these algorithms require a hierarchical structure in the network and prioritize the estimate of the higher-ranking node. Since time intervals defined by guaranteed bounds can be combined optimally and unambiguously by intersection, the interval-based approach does not require hierarchical structures.

We extend the results from previous sections in this chapter by showing that interval-based synchronization is not only resilient to node mobility, but actually even benefits from it in the average case. We will argue that this is not the case for time-estimate-based synchronization approaches that employ hierarchical structures such as clusters [EGE02] or trees [GKS03, SV03, vGR03], since mobility makes these structures prohibitively expensive to maintain. With

interval-based synchronization, nodes can use arbitrary communication partners, and therefore no such overhead exists. Our simulations show that random communication is a viable and competitive alternative to tree-based approaches.

We look at mobility in the context of external synchronization, using the system model and the algorithms from Section 3.3. We first discuss the various possible communication patterns in ad-hoc networks in Sections 3.8.1 and 3.8.2, showing that random communications are not only a good compromise, but may be the only feasible choice in scenarios with mobile nodes. In Section 3.8.3, we examine the effect of node mobility on interval-based synchronization, which our simulations show to be beneficial.

3.8.1 Communication patterns

To achieve synchronization, time information has to be propagated through the network. A communication pattern determines which nodes communicate at which time. Time information can be embedded in the messages sent by the application of the network or it can be transmitted in dedicated synchronization messages. In this section, we compare different communication patterns in a network with a single anchor node. There are two very different modes of operation:

- The network can be organized in a tree topology with the anchor node at the root. Each parent node synchronizes with all its children. We refer to this as the tree-based mode.

- Each node in the network synchronizes with a random node within its transmission range. We refer to this as the random-communications mode.

The tree-based mode is used by many recently proposed synchronization algorithms for ad-hoc networks, e.g. [GKS03, SV03, vGR03, KM04]. Various methods of distributed tree construction were discussed in [vGR03]. All these methods are expensive in terms of communication overhead, which increases when the nodes in the network become mobile and the tree has to be rebuilt frequently. In contrast, the random-communications mode does not suffer from this kind of overhead at all, since it does not use any topology information.

In this section, we compare the tree-based and the random-communications modes of interval-based synchronization in terms of the maximal uncertainty in the network. In addition, we compare the distribution of the frequency of communication among the nodes in the network. A high frequency of communication implies a high energy consumption and therefore a short node lifetime. To maximize network lifetime, it is desirable to have an equally distributed and small number of communications. In this section, we will show the following:

- In a breadth-first tree, the maximal uncertainty is small, but the frequency of communication is distributed very unevenly among the nodes.

- In a depth-first tree, the frequency of communication is distributed more evenly than in the breadth-first tree, but the maximal uncertainty is much larger.
- For a fixed total number of communications in the network, the random-communications mode achieves an only slightly larger maximal uncertainty than the breadth-first tree, while communications are distributed as evenly as in the depth-first tree.
- For a fixed maximal number of communications per node, i.e. a fixed minimal node lifetime, the random-communications mode achieves a strictly smaller uncertainty than either tree-based mode.

Specialized lower bounds

We now show how the uncertainty in the tree-based mode can be compared to the one achieved with random communications. To this end, we derive specific lower bounds on the uncertainty from (3.1) on page 57.

Lower bound in a tree

Let a tree with an anchor node at the root be given, and let f_C be the frequency of communications per node. If every parent node communicates with all its child nodes at least every $1/f_C$, then for all children of the anchor node, the real-time difference $(t - t_s)$ between any time t and the time t_s of the latest communication with the anchor node before time t is at most $1/f_C$. For an arbitrary node, the difference $t - t_s$ is at most $(\textit{hop distance to root})/f_C$. Thus, the worst-case uncertainty ΔT is bounded by

$$\Delta T \geq \frac{(\textit{maximal hop distance to root})}{f_C} \cdot \frac{2\hat{\rho}}{1-\hat{\rho}} \quad . \tag{3.7}$$

We see that the maximal uncertainty grows with the maximal hop distance to the root. Thus, a depth-first tree yields a large maximal uncertainty.

Lower bound for random communications

We determine the maximal real-time difference $(t - t_s)$ and thus the maximal uncertainty for random communications by simulation. In analogy to (3.7), we define a *pseudo hop distance* to the root node as

$$(\textit{pseudo hop distance to root}) = f_C \cdot \max_t \{t - t_s\} \quad .$$

Communication frequency per node

In both the tree-based and the random-communications mode, we assume that every non-anchor node initiates a communication with some node within its transmission range with a frequency of f_C. Thus, in a time interval of size Δt, $f_C \cdot \Delta t$ communications take place per node, each involving two nodes.

In a star topology, i.e. in the ideal breadth-first tree, all child nodes initiate a communication with the root node once in an interval of length $1/f_C$. The root node hence communicates n times (where n is the number of child nodes) within

each interval of length $1/f_C$. As a consequence, the root node consumes n times more energy than the child nodes.[7] In a chain, i.e., in the ideal depth-first tree, the root and the last node communicate once per $1/f_C$, and all other nodes communicate twice per $1/f_C$. Thus, the energy consumption is distributed almost evenly among all nodes. In a general tree, the maximal number of communications per $1/f_C$ is equal to the maximal node degree.

In the random-communications mode, the maximal number of communications per node and time unit is determined using simulation. The communications are distributed quite uniformly among the nodes due to the random selection of communication partners. A *pseudo node degree* can be determined by simulation by counting the maximal number of communications in which any node is engaged in an interval of length $1/f_C$.

3.8.2 Simulation with static nodes

We will now describe the simulations of networks with static nodes we performed to compare tree-based and random communication.

Setup

We placed 100 nodes randomly in a square with edge length 100. An anchor node was placed in the center of the square. For a transmission range between 10 and 80, breadth-first and depth-first trees were constructed. Figure 37 shows these trees for transmission ranges 15, 25, and 35.

In these trees, the maximal hop distance of any node to the root and the maximal degree of any node were determined. Using the same node locations, we simulated the random-communications mode during a real-time interval of $100/f_C$. From the obtained traces, we computed pseudo node degrees and pseudo hop distances. The maximal and minimal results from 50 runs of this experiment are shown in Figures 38 and 39.

Results

First, consider the maximal degree of any node in the network, shown in Figure 38. For the depth-first tree, the maximal degree is fairly low (≤ 4) and remains approximately constant for all transmission ranges. For the breadth-first tree, the maximal degree is small for a small transmission range and then quickly increases. At a range of $50 \cdot \sqrt{2} \approx 71$ $(x = 0.71)$, the root node can reach every node, and its degree is maximal, while all other nodes have degree 1.

Now, consider the maximal hop distance to the root node, shown in Figure 39. At a transmission range of 10 ($x = 0.1$), there are nodes that cannot communicate with any other node, the maximal distance is therefore not shown. This occurs up to and including a transmission range of 25. For the breadth-first tree, the maximal hop distance is fairly small (≤ 4 for a transmission range of at least 30). For the depth-first tree, the maximal hop distance is considerably larger and increases quickly (≥ 83 for a transmission range of at least 30).

[7] Note that in our model, communication is point-to-point. If broadcast is possible, the root node needs to communicate only once per $1/f_C$.

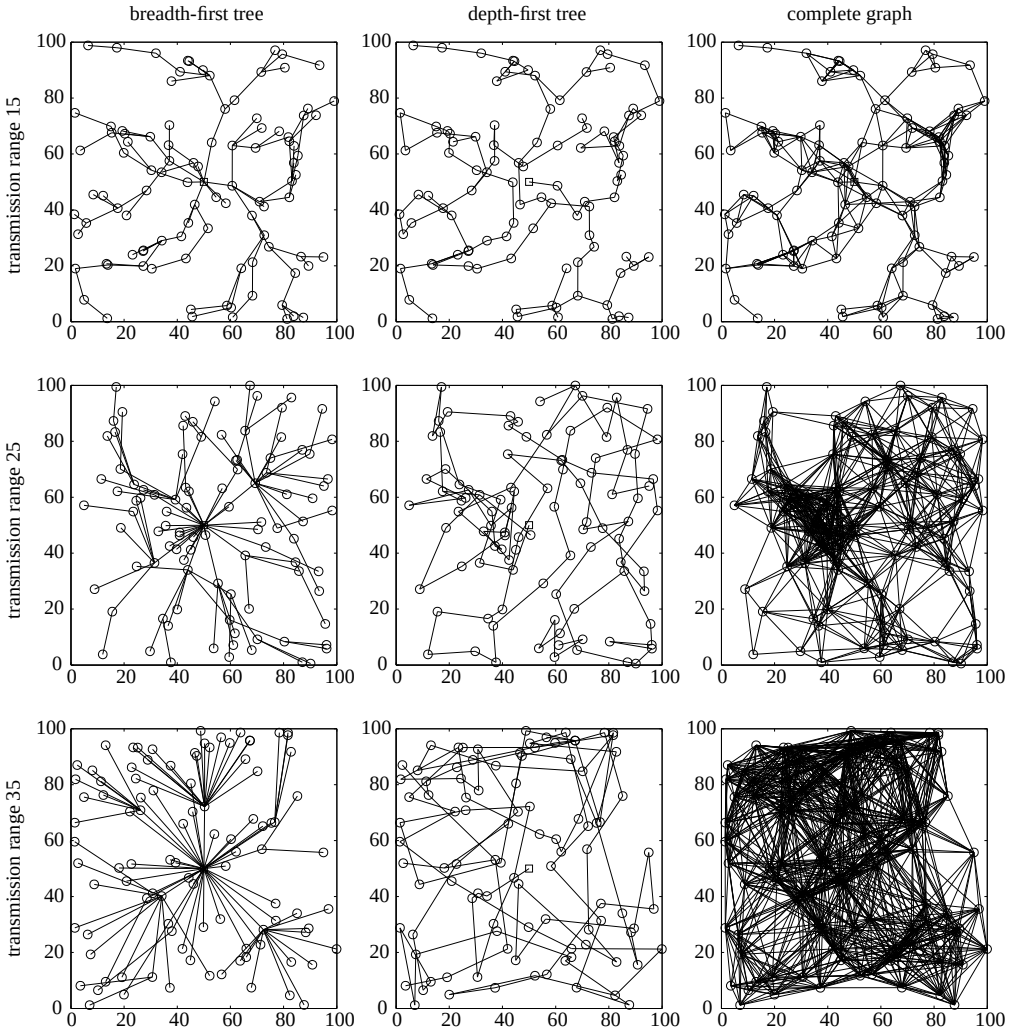

Fig. 37: 100 randomly placed nodes (depicted as circles), organized in a breadth-first tree (left column) and in a depth-first tree (middle column). The root (depicted as a square) is always in the center of the area. The right column shows the complete graph for a given transmission range.

In the random-communications mode, the communications are distributed as evenly among the nodes as in the depth-first tree, i.e. far better than in the breadth-first-tree. The maximal hop distance (and thus the worst-case uncertainty) of the random-communications mode is much better than that of the depth-first tree, but approximately 5 times worse than that of the breadth-first tree (≤ 20 for a transmission range of at least 30).

Exemplary values for maximal uncertainty ΔT and maximal communications per node for a transmission range of 0.3 times the area width and two different drift bounds $\hat{\rho}$ are given in Table 2 on page 93. The uncertainties ΔT scale with $\hat{\rho}$ and $1/f_C$, and there exist trade-offs between the three approaches.

Each row in Table 2 compares the different modes for a given *total* number of communications *in the network*. Figure 40 shows the maximal uncertainty for

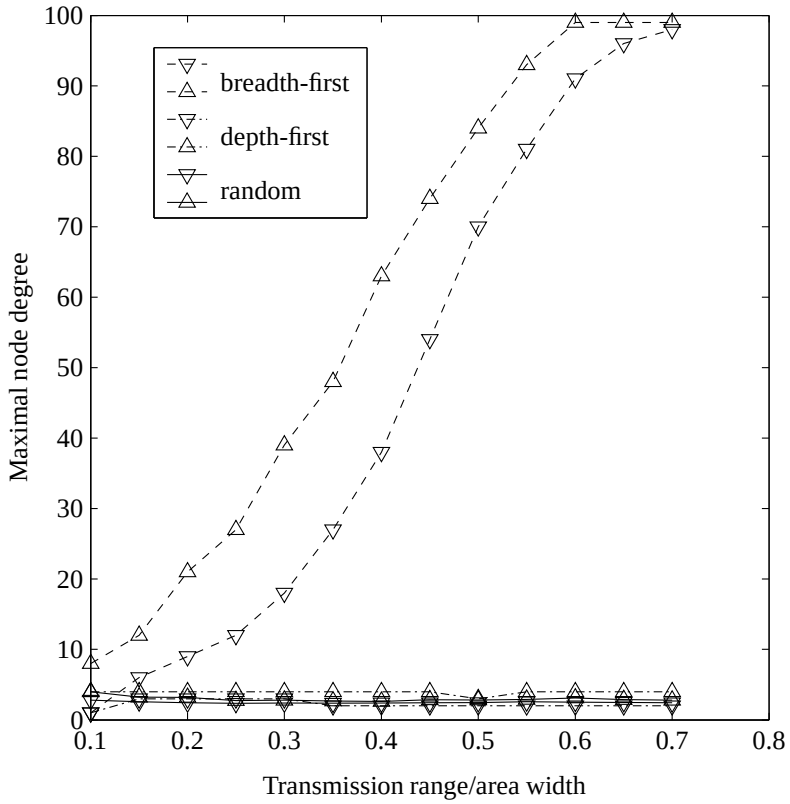

Fig. 38: Maximal node degree in breadth-first and depth-first trees constructed in networks of 100 randomly placed nodes as a function of the nodes' transmission range. The figure also shows pseudo node degrees determined from simulating the random-communications mode. The upward and downward triangles represent the maximal and minimal values from all simulation runs.

a given *maximal* number of communications *per node*, which is analogous to a given minimal lifetime of each node. Here, the random-communications mode achieves strictly better results than the tree-based modes.

Discussion

In this section, we have considered networks without node mobility and shown that in comparison to the tree-based mode, the random-communications mode offers a very good compromise between a small maximal uncertainty and fair distribution of communications among the nodes. For a given maximal communication frequency per node, it achieves a strictly smaller maximal uncertainty than the breadth-first and depth-first tree-based modes.

While the random-communications mode does not have any overhead at startup, the topology construction required in the tree-based modes is expensive. The authors of [vGR03] state that the communication overhead of breadth-first tree construction can be reduced to $10 \cdot n \cdot m^{1/2}$, where n is the number of nodes and m the number of edges in the network. The communication overhead of a distributed depth-first tree-construction algorithm is specified as $4 \cdot m$ in $4 \cdot n - 2$ rounds. In our simulations of the networks described above, the number of nodes

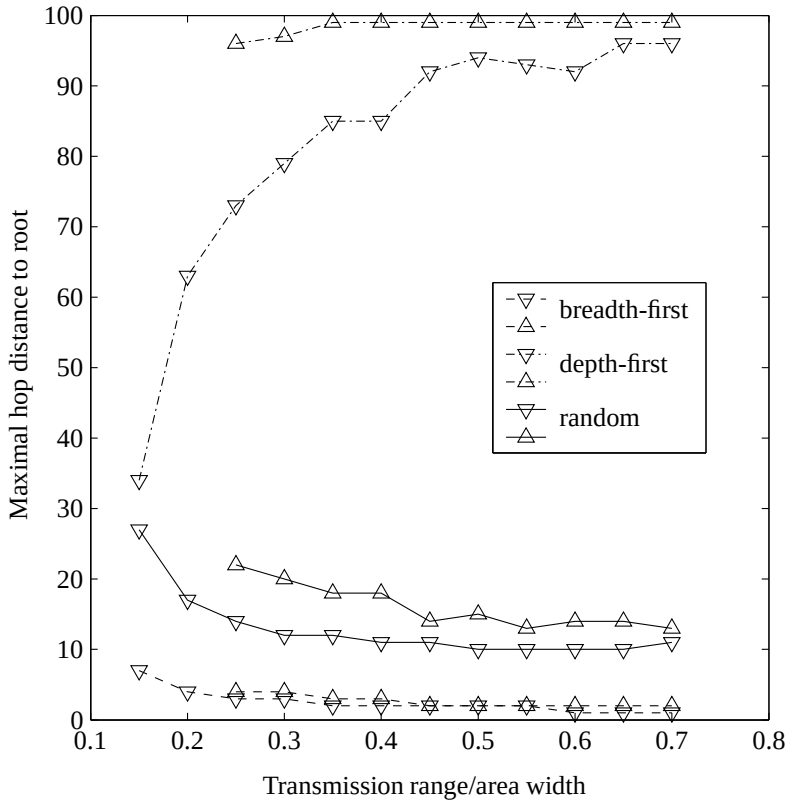

Fig. 39: Maximal hop distance to the root in breadth-first and depth-first trees constructed in networks of 100 randomly placed nodes as a function of the nodes' transmission range. The figure also shows pseudo hop distances determined from simulating the random-communications mode. The upward and downward triangles represent the maximal and minimal values from all simulation runs. The missing upward triangles on the left for transmission ranges below 30 ($x = 0.3$) are due to nodes not connected to the root.

is 100 and the average number of edges, assuming a transmission range of 30, is approximately 1100. Thus, the number of communications necessary for tree construction is approximately 13270 for breadth-first and 4400 for depth-first trees. This overhead becomes completely unbearable when the tree has to be reconstructed frequently due to node mobility.

3.8.3 Simulation with mobile nodes

We will now describe the scenarios with mobile nodes that we simulated and the performance measures we used to evaluate the results. To make our results comparable to those from Section 3.3, we chose similar parameters and merely added node mobility. The simulations were done using custom C++ programs.

Model and measures

We simulated sensor-network scenarios for a simulation time of 500 h, i.e. almost 21 days. The network contained 100 nodes that were distributed uniformly at random in a square area with edge length 10000. Their local clocks had a con-

f_C	Breadth-first tree		Depth-first tree		Random comm.	
	ΔT	cpn	ΔT	cpn	ΔT	cpn
Transmission range/area width $= 0.3$, $\hat{\rho} = 100$ ppm						
$1/\mathrm{h}$	2.8 s	$38/\mathrm{h}$	69.1 s	$4/\mathrm{h}$	14.4 s	$3/\mathrm{h}$
$1/\mathrm{min}$	48 ms	$38/\mathrm{min}$	1.2 s	$4/\mathrm{min}$	240 ms	$3/\mathrm{min}$
$1/\mathrm{s}$	800 µs	$38/\mathrm{s}$	19 ms	$4/\mathrm{s}$	4 ms	$3/\mathrm{s}$
Transmission range/area width $= 0.3$, $\hat{\rho} = 1$ ppm						
$1/\mathrm{h}$	28 ms	$38/\mathrm{h}$	691 ms	$4/\mathrm{h}$	144 ms	$3/\mathrm{h}$
$1/\mathrm{min}$	480 µs	$38/\mathrm{min}$	12 ms	$4/\mathrm{min}$	2.4 ms	$3/\mathrm{min}$
$1/\mathrm{s}$	8 µs	$38/\mathrm{s}$	190 µs	$4/\mathrm{s}$	40 µs	$3/\mathrm{s}$

Tab. 2: Maximal uncertainty ΔT and maximal communication frequency for a transmission range of $0.3 \cdot$ area width and two different constants $\hat{\rho}$. The uncertainty ΔT scales with $\hat{\rho}$ and $1/f_C$. The acronym "cpn" indicates "communications per node".

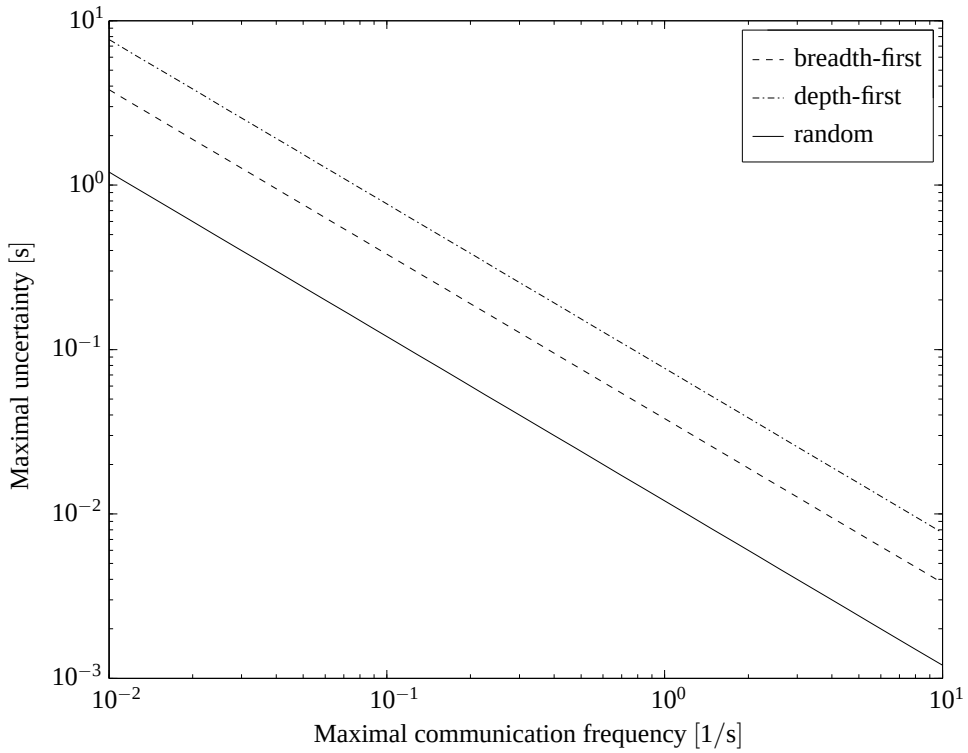

Fig. 40: For an equal maximal communication frequency per node, the random-communications mode achieves a strictly smaller maximal uncertainty than the breadth-first and depth-first tree-based modes.

stant drift rate between -100 ppm and $+100$ ppm. A number $N_A \in \{5, 10, 20\}$ of the 100 nodes were anchor nodes. In each run, all nodes had the same transmission range; between runs, it was varied between 0.1 and 0.5 times the width of the simulation area. Sensor nodes communicated with other sensor nodes at an average frequency of $f_C = 2/\text{h}$. Anchor nodes communicated with sensor nodes at a smaller average frequency $f_A = 0.02/\text{h}$.

In our mobility model, both anchor and non-anchor nodes can move according to a model similar to the random-waypoint model [JM96]: We first specify the number $l \geq 1$ of locations per node. For each node, we choose uniformly at random $l-1$ real times $t_1, \ldots, t_{l-1}$ within the simulation time frame and l locations $L_0, \ldots, L_{l-1}$. We set $t_0 := 0$. At real time t_i, the node is at location L_i. For any two consecutive locations i and $i+1$, the node moves with constant speed on a direct line from location i to location $i+1$. There is no pause at locations.

The number of locations can be given separately for anchor and non-anchor nodes. We let anchor nodes have only one location and placed them on a regular grid that maximized their coverage of the simulation area. This was motivated by the fact that a limited number of relatively powerful, immobile anchor nodes can be expected to be placed optimally with respect to area coverage.

The performance measure we used was the time uncertainty; we evaluated it after every communication and took the average over all communications, excluding infinite time uncertainties. We thus produced the average time uncertainties for the algorithm IM and the BP-ISA. From these values, we computed the improvement of the BP-ISA. Figures 41 to 43 give all these results as percentages and in milliseconds. For every data point, 300 runs were performed. The standard deviations for the percentage improvements are shown in Table 3.

anchors	standard deviation					
	transm. range 500		transm. range 1500		transm. range 2500	
	$x = 1$	$x > 1$	$x = 1$	$x > 1$	$x = 1$	$x > 1$
5	0.96	4.63–6.16	4.29	2.32–3.04	5.88	3.14–4.08
10	1.23	3.52–4.68	3.74	1.79–2.66	4.79	2.47–3.56
20	1.86	2.75–4.15	3.39	1.34–2.48	4.21	1.90–2.78

Tab. 3: Standard deviations for the improvement percentage of the BP-ISA over the algorithm IM depicted in Figures 41 to 43. The values for no mobility ($x = 1$) differ considerably from those for mobility ($x > 1$).

Discussion

We draw the following conclusions from Figures 41 to 43:

- The uncertainties become smaller when the mobility becomes larger (for $x > 1$). This effect diminishes with increasing transmission range, since the improvement in connectivity becomes smaller.
- Analogously, the uncertainties become smaller when the range increases. This effect is larger for low mobility than for high mobility.

- For transmission range 500, there is a sharp jump between $x = 1$ and $x = 5$ in the graphs showing the uncertainty, and the uncertainty for $x = 1$ is much smaller than that for $x = 5$, suggesting that no mobility leads to smaller uncertainty. The reason for this is that at $x = 1$, i.e. without mobility, most of the nodes are not connected to the anchor. Their uncertainty thus is infinite and is not considered. The few nodes that are connected to the anchor are only one hop away from it. Their uncertainty thus is very small. As the transmission range grows larger, this effect diminishes.

Our simulation results suggest that mobility has a positive overall effect on the synchronization quality, both for the algorithm IM and the BP-ISA. This effect is due to the increase in connectivity (which for large transmission ranges is small); there does not seem to be any negative effect such as disruption of back paths used by the BP-ISA.

3.9 Summary

In this chapter, we presented a zero-delay model for time synchronization in mobile ad-hoc networks and used it to identify the worst and the best case in terms of achievable time uncertainty. We showed that the algorithm IM from [MO83] is worst-case-optimal and proposed an improved algorithm which takes advantage of the typical drift diversity of the nodes' clocks. We illustrated by simulation that in the average case, our algorithm significantly outperforms the algorithm IM from [MO83] at a moderate increase in computation and memory cost.

Next, we turned to internal synchronization and improved the algorithm from [Röm01], obtaining better synchronization with less computation and no additional communication or memory. We then presented algorithms for internal and external synchronization which use all the data that can be obtained for a given communication pattern, at the expense of additional computation, communication and memory. This lead us to analyze the trade-off between synchronization quality and required resources.

Finally, we showed that node mobility puts no burden on interval-based synchronization, but actually increases its quality. Furthermore, we argued that synchronization algorithms that employ hierarchical communication structures are bound to suffer from prohibitive overhead if nodes are mobile. We showed that for the interval-based approach, unstructured communication offers a very good compromise between a small maximal uncertainty and fair distribution of communication among the nodes.

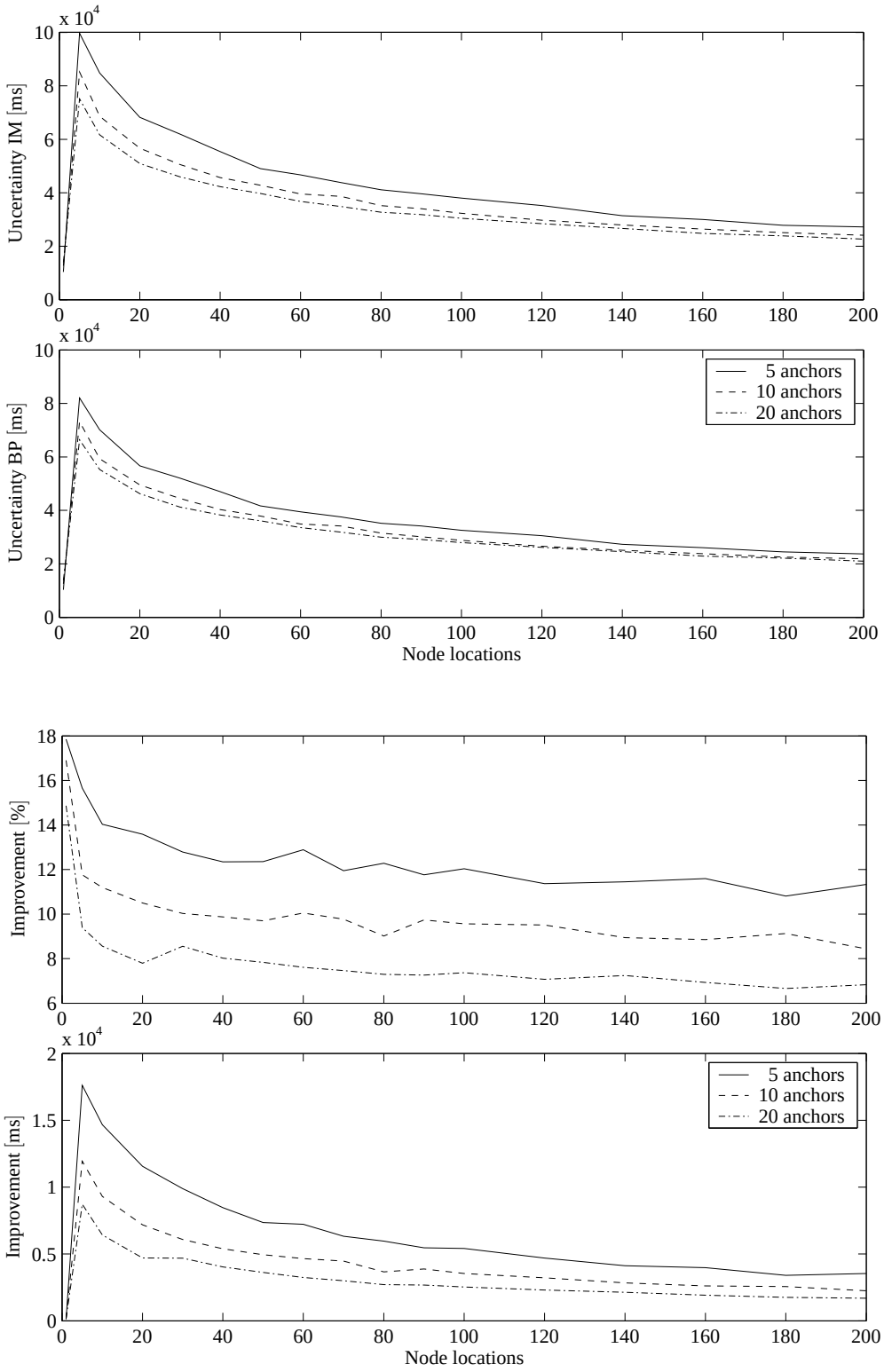

Fig. 41: Simulation results for transmission range 500. The uncertainties of the two algorithms and the improvement of the BP-ISA over the algorithm IM are shown as a function of the number of locations each node had during the simulation time, i.e. as a function of node mobility.

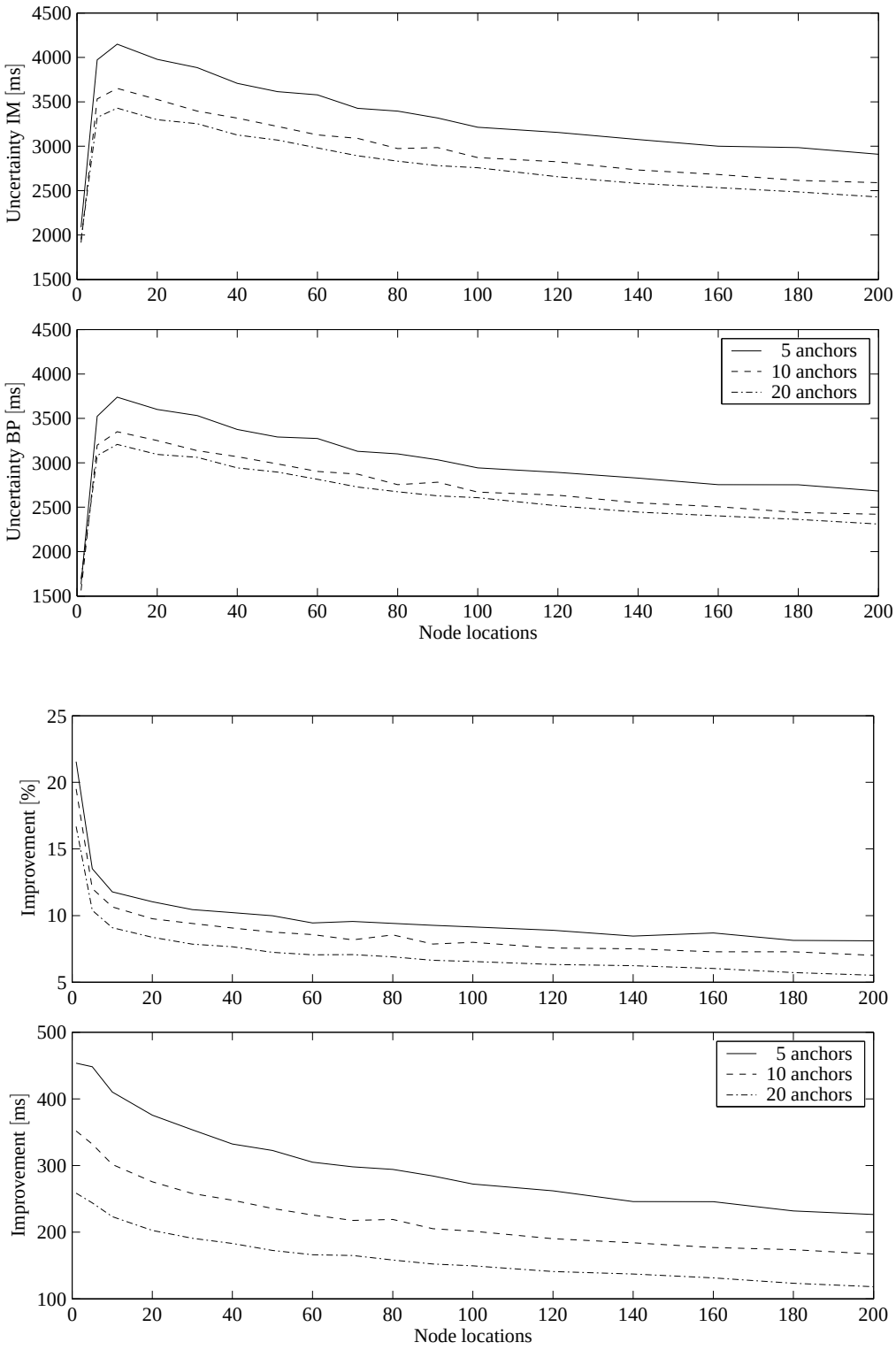

Fig. 42: Simulation results for transmission range 1500. The uncertainties of the two algorithms and the improvement of the BP-ISA over the algorithm IM are shown as a function of the number of locations each node had during the simulation time, i.e. as a function of node mobility.

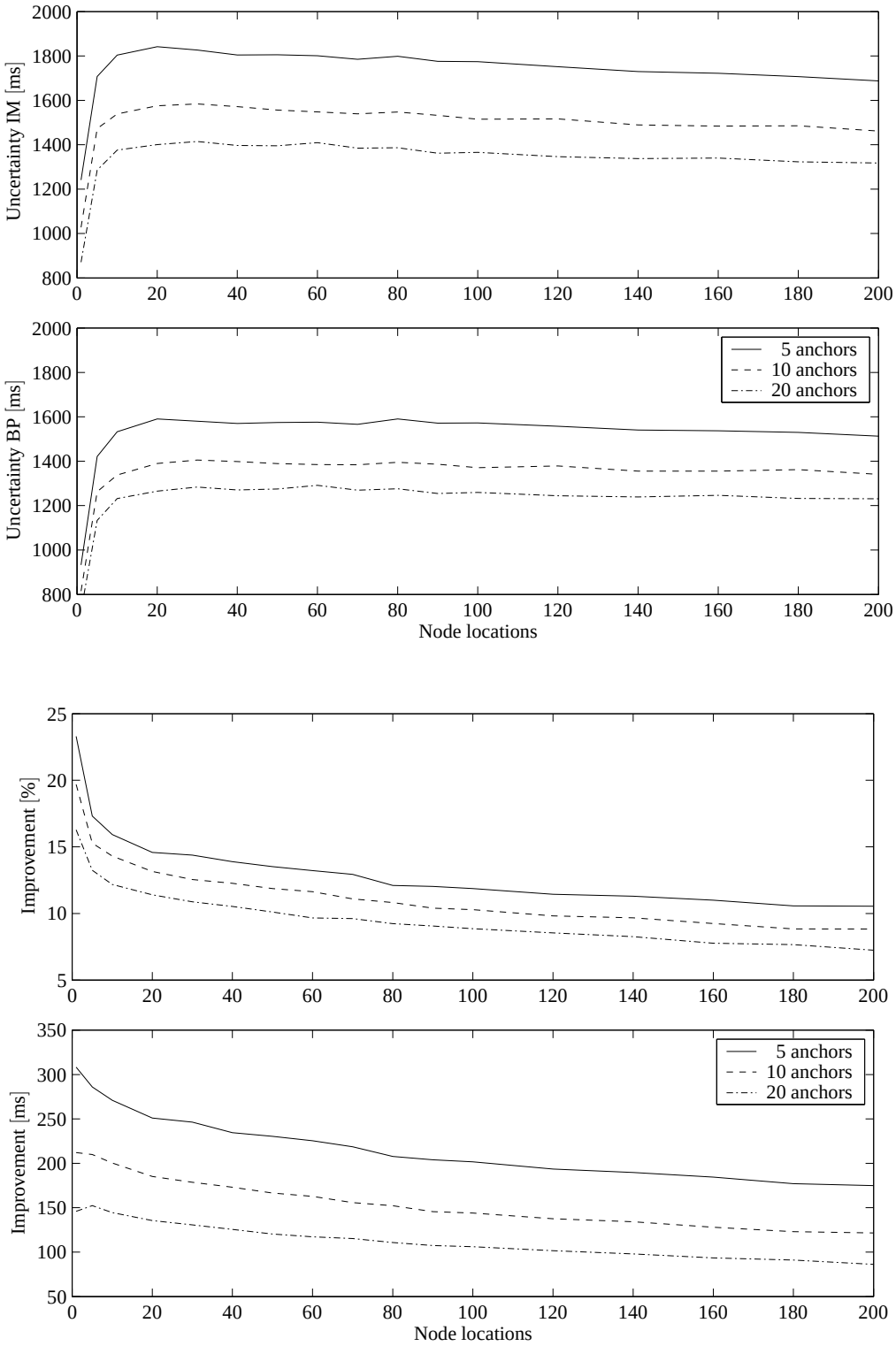

Fig. 43: Simulation results for transmission range 2500. The uncertainties of the two algorithms and the improvement of the BP-ISA over the algorithm IM are shown as a function of the number of locations each node had during the simulation time, i.e. as a function of node mobility.

4

Gradient Clock Synchronization in Wireless Sensor Networks

4.1 Introduction

Gradient clock synchronization [FL04] can be seen as a special case of internal synchronization. The gradient property requires the difference between any two network nodes' clocks to be bounded from above by a non-decreasing function of their distance. The distance between two nodes can be defined for example as their Euclidean distance, as their hop distance in the network, or as the message-delay uncertainty between them. For the successful completion of actions by groups of sensor nodes, the gradient property is both necessary and sufficient. We illustrate this with two examples:

- For energy-efficient operation, the duty cycle of a sensor node's communication circuits has to be synchronous to that of its communication partners, i.e. of the nodes in its one-hop neighborhood. These are the nodes with the smallest distance.
- When deriving the speed of an object from two nodes' observations, the total error is proportional to the quotient of the synchronization error and the Euclidean distance between the nodes. For a given maximum total error, faraway nodes are allowed to be more loosely synchronized than nodes that are close to each other [FL04].

In [FL04], a lower bound on the error of gradient clock synchronization was given. The system model in which the bound was derived differs considerably from the one we use in this thesis. In this chapter, we show that an analogous lower bound exists also in our model.

The rest of the chapter is organized as follows: We present an overview of our new results and related work in Section 4.2. In Section 4.3, we provide our system model. We then derive the lower bound for gradient clock synchronization in this model in Section 4.4.

This chapter is based on [MT05a, MT05b].

4.2 New Results and Related Work

Synchronization is hindered by two factors: clock drift and message-delay uncertainty. Time information is degraded by the drift of imperfect clocks (since nodes cannot precisely measure time intervals), and it cannot be communicated without loss due to delay uncertainties (since the age of received time information cannot be determined accurately). Upper bounds on the drift and the delay uncertainty allow us to derive lower bounds on the synchronization error between two network nodes. For instance, the worst-case clock difference between two nodes with delay uncertainty D is in $\Omega(D)$ [LL84], i.e. for any algorithm, a scenario can be constructed where the clock difference is in $\Omega(D)$. The central theorem of [FL04] states that in a network with maximal delay uncertainty D, the clock difference of two nodes with constant delay uncertainty is in $\Omega(\frac{\log D}{\log\log D})$. This implies that the clock difference grows as D grows although the delay uncertainty between the nodes remains constant.

In the system model of [FL04], nodes can communicate with unbounded frequency and are equipped with hardware clocks with bounded drift. The delays and drifts are unknown to the nodes and hence to any synchronization algorithm. To derive a lower bound on the error, it is assumed that an adversary can control the delays and drifts. The communication pattern, the delays, and the nodes' clock drifts constitute an execution trace. For a given trace, the adversary constructs another trace at the end of which two given nodes have a larger clock difference than in the original trace. This process can be repeated sufficiently often for the lower bound to be attained. During the iterations, the clock difference increases faster than any synchronization algorithm can reduce it without violating the gradient property.

In this chapter, we use a different system model that we consider more appropriate for wireless sensor networks. The characteristics of our model and of the previous model are summarized in Figure 4. Clock drifts are bounded in both models, but we assume that the communication frequency is bounded, i.e. that there is no arbitrarily high amount of communication between nodes, since this would lead to a prohibitive energy consumption. The synchronization service uses only the communication that takes place anyway for achieving the overall goal of the sensor network; the time information is sent piggyback with the application data [PSR94, AHR96]. Therefore, the communication pattern is typically not known to the synchronization algorithm. Examples of such sensor networks are those in which environmental data is collected on a regular basis

	previous model	our model
clock drift	bounded	bounded
communication frequency	unbounded	bounded
message-delay uncertainty	bounded	0
message delay	bounded	0

Tab. 4: Overview of the characteristics of the system model from [FL04] and of our model.

but communicated only sporadically, e.g. when time-critical data is recorded, when a master node explicitly requests the data, or when the solar cells of the node are providing sufficient energy for communication.

As discussed in Section 3.2, the influence of delay uncertainty and drift can be studied separately [BMT04, MBT04], and the drift's effect is dominant in networks with infrequent communication. We hence assume delay uncertainties to be negligible, and eliminate the delays themselves from our analysis, assuming communication to occur in zero time.

In order to be able to derive any bounds, we obviously have to make some quantitative assumption about the communication frequency. We assume that a node communicates at least once every d time units with each of its neighbors. In practice, this could be a bound on the communication frequency determined by the application. The adversary can modify the communication pattern by shifting the times at which communication events occur. He can also change the clock drifts. As there are no delays, we use the hop distance between nodes as a distance measure instead of the delay uncertainty. Thus, the network diameter is the maximal hop distance in the network.

We show that an analogous lower bound as in the previous model exists also in our model. This is not obvious: In the previous model, the bound is based on delay uncertainties, which do not exist in our model. Our bound is based on the parameter d, which leads to an upper bound on the delay of a message in the network. This may seem an isomorphism between the two models, where the bound on communication frequency replaces the bound on delays. But note that when the adversary in our model shifts the time at which a communication event occurs, he also has to modify the clock drifts of both nodes involved in order to make the modified trace indistinguishable from the original. In the previous model, the adversary that modifies the delay of a message need only adjust the clock drift of either the sending or the receiving node. This means that the adversary is more powerful in the previous model. Hence there is no isomorphism between the two models, and it is not a priori clear that a comparable bound exists in our model.

For the clock difference $|L^j(t) - L^i(t)|$ between two neighboring nodes v_i, v_j in a chain of n nodes and maximal clock drift $\hat{\rho}$, we derive the lower bound

$$|L^j(t) - L^i(t)| \geq \frac{\hat{\rho} d}{8(1+\hat{\rho})} \frac{\log(n-1)}{\log\left(\frac{8(1+\hat{\rho})}{\hat{\rho}} \log(n-1)\right)} \quad .$$

Analogously to the bound from [FL04], our bound implies that the clock difference of neighboring nodes increases with the network diameter or chain length $n-1$. The bound increases with increasing d and $\hat{\rho}$.

4.3 System Model

In our system model, each node v_i in the network uses its hardware clock to measure time differences and ultimately to compute its logical clock L^i. Node v_i executes a local synchronization algorithm (LSA) that satisfies the following properties:

- It is deterministic, has access to unbounded memory, and executes instantaneously.
- It exchanges information with some node v_j via a communication event.
- It can read the local time from the hardware clock at any time.
- It determines the logical clock L^i.
- In particular, it does not have access to the real time or to the hardware clock's drift, and it has no influence on the occurrence of communication events.

Def. 18: **(Hardware clock)** *A hardware clock is a physical device, typically a counter, modeled as a function $H : \mathbb{R} \to \mathbb{R}$. We use the notation $h_k = H(t_k)$ to denote the hardware-clock reading or* local time *h_k of an event k that occurs at real time t_k. The drift of a hardware clock is*

$$\rho(t) = \frac{\mathrm{d}H(t)}{\mathrm{d}t} - 1 \ .$$

It is constrained by $-\hat{\rho} \leq \rho \leq \hat{\rho}$ for some constant $\hat{\rho} \geq 0$.

Def. 19: **(Network)** *A network $N = \{v_i : 0 \leq i < n\}$ is a set of n nodes. Every node v_i has a hardware clock H^i. Communication between two nodes is modeled as a communication event that enables the exchange of information in zero time. Formally, it is a tuple (v_i, v_j, t_k), meaning that nodes v_i and v_j communicate at real time t_k by event k.*

Def. 20: **(Scenario and trace)** *A scenario $S = (N, \{(v_i, v_j, t_k)\})$ is a communication graph with vertices that correspond to nodes in the network N and edges (v_i, v_j) with weight 1 that correspond to communication events (v_i, v_j, t_k). A trace T is a scenario augmented with local times h_k^i and h_k^j of the nodes v_i and v_j involved in the communication event k.*

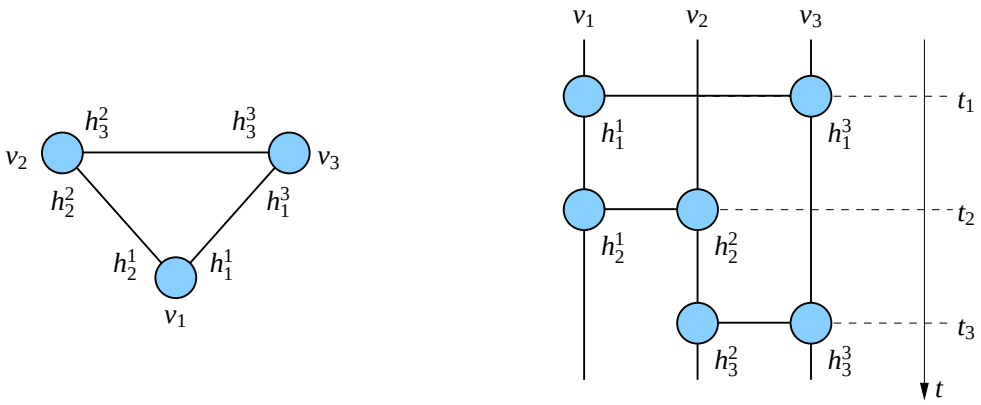

Fig. 44: Communication-graph and event-chart representation of a trace.

A trace can be depicted as a communication graph or as an event chart, see Figure 44.

Def. 21: **(Gradient property (GP))** *Let any non-decreasing function $f : \mathbb{R}^+ \to \mathbb{R}^+$ be given. An LSA satisfies the f-gradient property (f-GP), if for any trace and for all $i, j \in \{0, n-1\}$*

$$|L^j(t) - L^i(t)| \leq f(s_{ij}) \ ,$$

where L^i denotes the logical clock of node v_i, and s_{ij} denotes the length of the shortest path between nodes v_i and v_j in the communication graph according to Definition 20.

The synchronization problem is characterized by the following properties:

- All hardware clocks start with local time $H^i(0) = 0$.
- Each node communicates with each of its neighbors at least every d time units, i.e. if v_i and v_j are neighbors, then they communicate at least once in every interval $[t, t+d]$ for all $t \geq 0$.
- All logical clocks run with a rate of at least $1/2$:

$$L^i(t + \Delta t) - L^i(t) \geq \Delta t/2 \ .$$

- The communication graph is connected.

4.4 Lower Bound

For the rest of the chapter, we assume a simple communication graph, namely a chain of n nodes: v_i and v_j are neighbors iff $|j - i| = 1$. We furthermore use $\gamma = \frac{\hat{\rho} d}{1+\hat{\rho}}$ for given $\hat{\rho}$ and d.

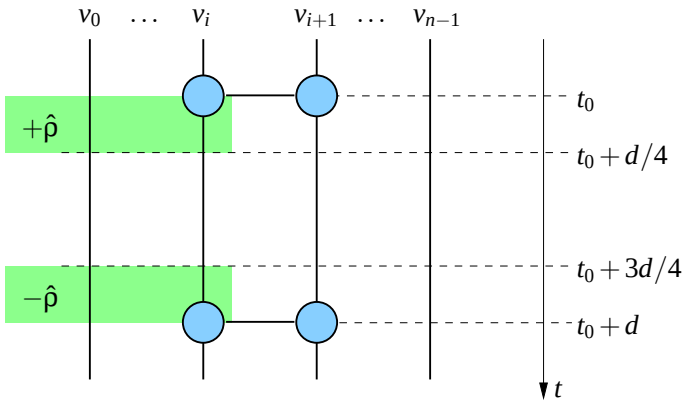

Fig. 45: Modification of clock drifts that leads to limits on the increase of the logical clock.

In this section, we will proceed as follows: Theorem 22 and Corollary 23 show that under certain conditions, a node's logical clock cannot change too fast without violating the gradient property. In Theorem 24, we show that for a given trace T, we can construct an indistinguishable trace T' which is shorter than T and at the end of which two distinct nodes have a larger clock difference than at the end of T. Corollary 25 uses this result to show that the error between two nodes is also lower-bounded by a linear function of their distance. The claim of Theorem 26 is similar to that of Theorem 24, with the addition that the constructed trace fulfills the prerequisites of Theorem 26 and can hence be used as input for another iteration. This leads directly to Theorem 27, where the lower bound on the clock difference of two neighboring nodes is derived by showing that a sufficient number of iterations can be performed.

Thm. 22: *Let an LSA satisfying the f-GP be given, and let all hardware clocks have drift $\rho = 0$. Let a node communicate with a neighbor at t_0 and the next time at $t_0 + d$. Then in any time interval of length $\hat{\rho}d/4$ that is enclosed in $[t_0 + d/4, t_0 + 3d/4]$, the node's logical clock cannot change by more than $2f(1)$.*

Proof: In the following, we consider the node v_i, its neighbor v_{i+1}, and two traces, T and T'. In T, all nodes have drift $\rho = 0$. In T', an adversary increases the drift of nodes v_0 to v_i by $\hat{\rho}$ in $t \in [t_0, t_0 + d/4]$ and decreases it by the same amount in $t \in [t_0 + 3d/4, t_0 + d]$, see Figure 45.

First, we will determine the time interval τ according to Figure 46. It is the interval by which v_i's hardware clock is shifted in T' with respect to T. We denote node v_i's local times at time $t_0 + d/4$ in T and T' as h_1 and h_2. Thus, $\tau = h_2 - h_1 = \hat{\rho}d/4$.

Now, we determine by how much the logical clock of v_i can increase within a real-time interval of size τ. We denote the difference between the logical clocks of v_i and v_{i+1} in trace T at time $t_0 + d/4$ as ΔL_1. If v_i increases its logical clock by ΔL between h_1 and h_2, then the difference between the logical clocks of v_i

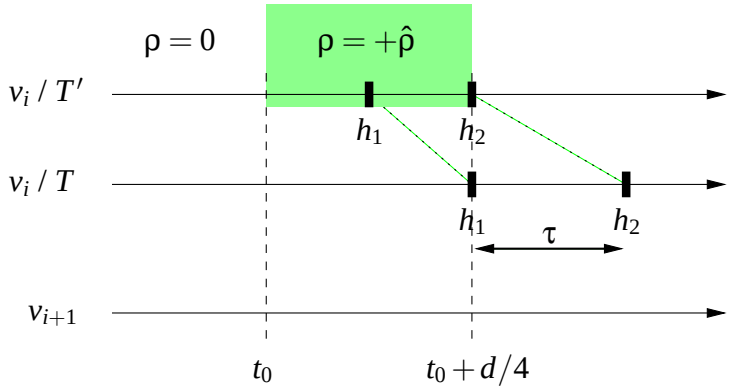

Fig. 46: Detailed view of the time shift in Figure 45. The clock of v_i is accelerated in trace T'.

and v_{i+1} in trace T' at time $t_0 + d/4$ is $\Delta L_2 = \Delta L_1 + \Delta L$. With $|\Delta L_1| \leq f(1)$ and $|\Delta L_2| \leq f(1)$, we obtain $|\Delta L| \leq 2f(1)$. The same argument holds for all intervals of size τ within $[t_0 + d/4, t_0 + 3d/4]$. ■

Cor. 23: *Let an LSA that satisfies the f-GP be given, and let all hardware clocks have drift $\rho = 0$. Let the time difference between two communications of a node to the same neighbor be d, and that between two arbitrary communications at least $d/2$ [1]. Then in any time interval of length $\hat{\rho}d/4$, the node's logical clock cannot change by more than $2f(1)$.*

Proof: The corollary follows directly from Theorem 22. Let us suppose that node v_i communicates with v_{i+1} at kd and with v_{i-1} at $d/2 + kd$ for $k \geq 0$. Then the change of the logical clock of v_i is bounded in $[d/4 + kd, 3d/4 + kd]$ because of the communication with v_{i+1}, and in $[d/4 + d/2 + kd, 3d/4 + d/2 + kd] = [3d/4 + kd, d/4 + (k+1)d]$ because of the communication with v_{i-1}. Hence, it is bounded at all times. ■

Thm. 24: *Let a trace T, two nodes v_i and v_j, $j > i$, and a time interval $I = [t_0, t_1]$, $t_1 = t_0 + \frac{1}{2}(j-i)d$ be given. Let the time difference between two communications of a node to the same neighbor be d, and that between two arbitrary communications at least $d/2$. In addition, let us assume that all drifts ρ are 0 in I and that at time t_1, we have $L^j(t_1) - L^i(t_1) = \lambda$.*

Then there exists another trace T' that is identical to T before t_0 and that at real time $t_2 = t_1 - (j-i)\frac{\hat{\rho}d}{2(1+\hat{\rho})}$ satisfies

$$L^j(t_2) - L^i(t_2) \geq \lambda + (j-i)\frac{\hat{\rho}d}{4(1+\hat{\rho})} \quad .$$

[1] Note that this, together with the previous condition, implies it to be exactly $d/2$.

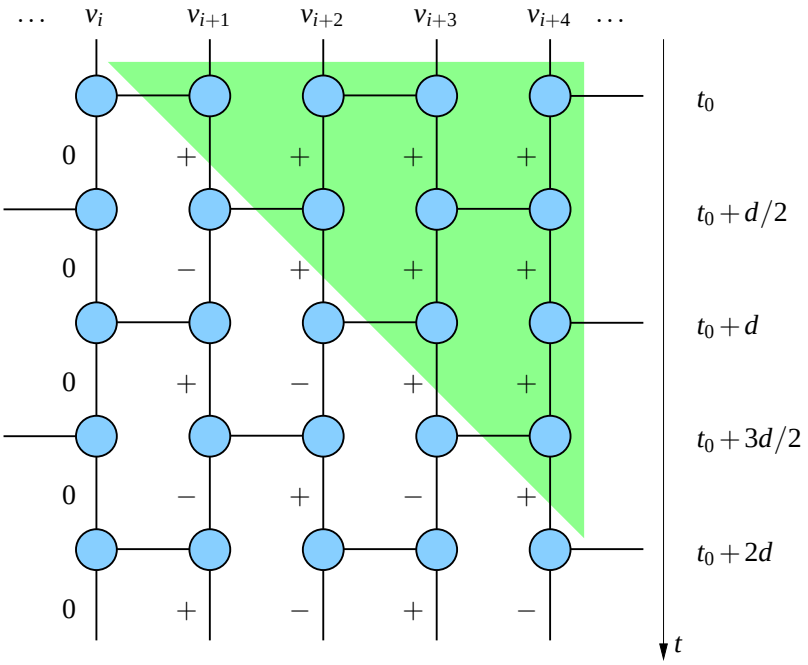

Fig. 47: Exemplary scenario for Theorem 24. Note the triangle of fast clocks.

Proof: An exemplary scenario is depicted in Figure 47, where $j = i+4$. In trace T, all drifts ρ are 0. In trace T', the drifts of all nodes are 0 until t_0. For nodes v_0 to v_i, the drifts remain 0 throughout the whole scenario; the other nodes' drifts change as depicted in Figure 47, where "0" denotes $\rho = 0$, "+" denotes $\rho = +\hat{\rho}$, and "−" denotes $\rho = -\hat{\rho}$.

The adversary changes the speed of a hardware clock in T' at the same local time at which a communication took place in trace T. Moreover, he also shifts the real time at which the communication takes place in T', thus making T' indistinguishable from T. It is clear from Figure 47 that the scheme of changing drifts is consistent with the scenario described in the theorem.

Now, we can calculate the time difference $\tau = t_1 - t_2$ between the communication event at node v_j at time $t_1 = t_0 + \frac{1}{2}(j-i)d$ in trace T and at time t_2 in trace T', see Figure 48. Between times t_0 and t_1, $\rho = 0$ in T, and hence the local time of v_j advanced by $\Delta h = t_1 - t_0$. The difference in local times between the two events is identical in T'. As $t_2 - t_0 = \frac{t_1 - t_0}{1+\hat{\rho}}$, we obtain $\tau = (j-i)\frac{\hat{\rho}d}{2(1+\hat{\rho})}$.

Finally, we denote as $\Delta L_1 = \lambda$ the (positive) difference between the logical clocks of v_j and v_i at time t_1 in T. As the speed of the logical clock is $\geq 1/2$, the difference ΔL between the logical clocks of v_i at t_2 and t_1 is $\Delta L \geq \tau/2$. Since the reading of v_j's logical clock at t_1 in T is identical to that at t_2 in T', we obtain $\Delta L_2 \geq \Delta L_1 + \tau/2$, where ΔL_2 denotes the difference of the logical clocks at t_2 in T'. ■

The following corollary shows that the error between two nodes is also lower-bounded by a linear function of their distance.

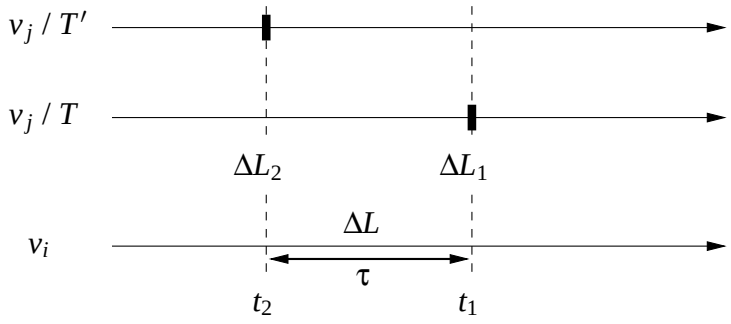

Fig. 48: Detailed view of the time shift in Figure 47. The clock of v_j is accelerated in trace T'.

Cor. 25: *For any LSA, there exists a trace such that at some real time t, two nodes v_i and v_j with distance s_{ij} have a difference of their logical clocks $|L^j(t) - L^i(t)| \geq s_{ij}\frac{\gamma}{8}$.*

Proof: Let us suppose that for some t_1, $L^j(t_1) - L^i(t_1) = \lambda$. We assume $|\lambda| < s_{ij}\frac{\gamma}{8}$, otherwise we are done. Then according to Theorem 24, there exists an indistinguishable trace with

$$L^j(t_2) - L^i(t_2) \geq \lambda + s_{ij}\frac{\gamma}{4} > -s_{ij}\frac{\gamma}{8} + s_{ij}\frac{\gamma}{4} = s_{ij}\frac{\gamma}{8} \ .$$

■

Thm. 26: *Given some $k \geq 0$, $n > n_k > 0$ and t_0. Given a trace T with the following properties:*

- *The trace is defined in $I = [t_0, t_1]$, where $t_1 = t_0 + \frac{1}{2}n_k d$.*
- *The drifts ρ of all nodes are 0 in I.*
- *The time difference between two communications of a node to the same neighbor is d, and that between two arbitrary communications is at least $d/2$.*
- *The difference of the logical clocks between some nodes v_{i+n_k} and v_i at real time t_1 is bounded by $L^{i+n_k}(t_1) - L^i(t_1) \geq kn_k\frac{\gamma}{8}$.*

For a given n_k, we define $n_{k+1} = \frac{n_k\gamma^2}{64df(1)} < n_k$. Then for any LSA, there is another trace ending with a trace T', and T' has the following properties:

- *The trace is defined in $I = [t_2, t_3]$, where $t_2 = t_1 - n_k\frac{\gamma}{2}$ and $t_3 = t_2 + \frac{1}{2}n_{k+1}d$.*
- *The drifts ρ of all nodes are 0 in I.*
- *The time difference between two communications of a node to the same neighbor is d, and that between two arbitrary communications is at least $d/2$.*

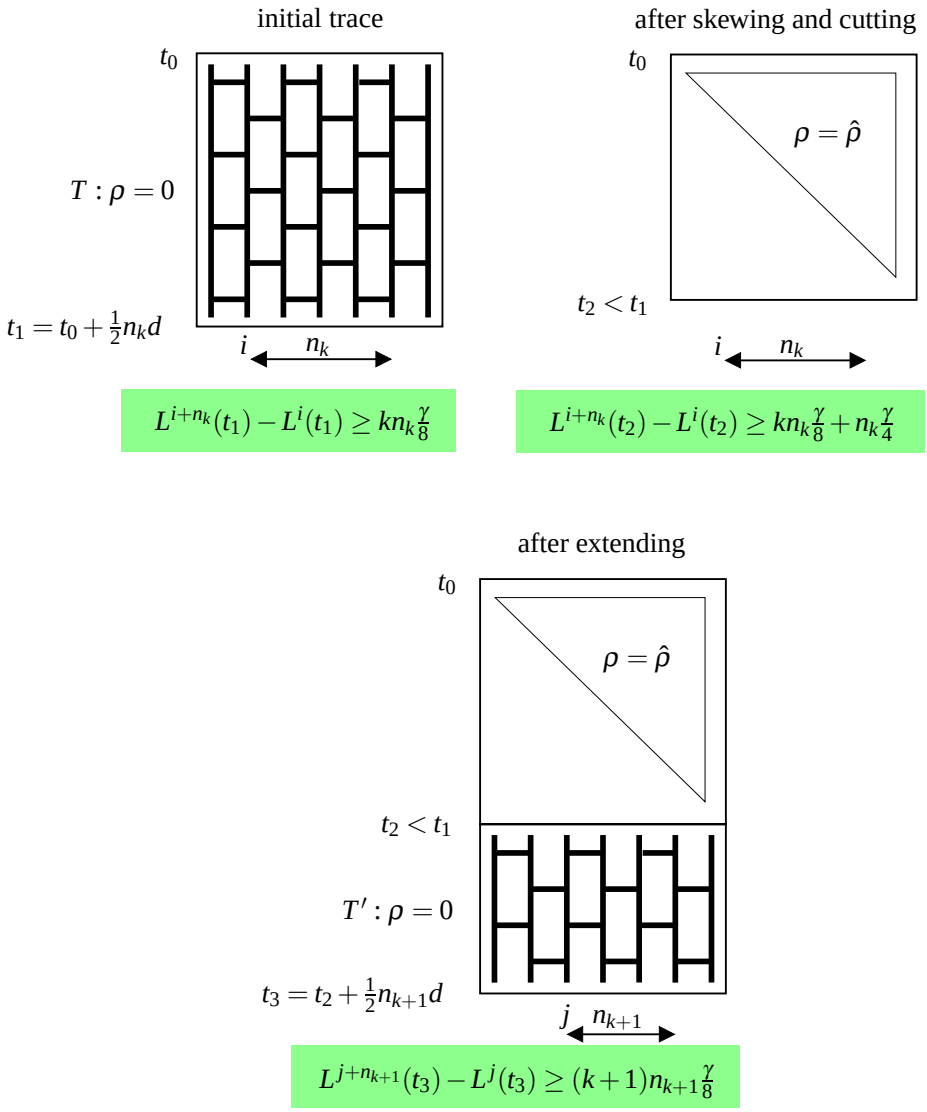

Fig. 49: One step of the adversary's construction: The initial trace T is first skewed and cut. Then, another trace is appended and becomes the new "initial trace".

- *The difference of the logical clocks between some nodes $v_{j+n_{k+1}}$ and v_j is bounded by $L^{j+n_{k+1}}(t_3) - L^j(t_3) \geq (k+1) n_{k+1} \frac{\gamma}{8}$.*

Proof: The stages of one step in the adversary's construction of a new trace are depicted in Figure 49. The initial trace satisfies the conditions of Theorem 24. Therefore, the adversary can construct another trace that yields

$$L^{i+n_k}(t_2) - L^i(t_2) \geq k n_k \frac{\gamma}{8} + n_k \frac{\gamma}{4}$$

as the LSA cannot be aware of the change of the trace. We have $t_2 = t_1 - n_k \frac{\gamma}{2}$. The adversary now cuts the constructed trace at t_2 and appends trace T'.

This cut obviously does not change the difference in the logical clocks at t_2, but the LSA may be aware of this change during T'. On the other hand, T' is

of the form we need in order to apply Corollary 23. The length of this trace is $\frac{1}{2}n_{k+1}d$. According to Corollary 23, a logical clock cannot change by more than $2f(1)$ in time $\hat{\rho}d/4$. Of course, it also cannot change more in a smaller interval $\gamma/4$. The logical clocks of both v_i and v_{i+n_k} can change at most by the same amount. Therefore, their difference cannot change by more than $4f(1)$ in time $\gamma/4$. As a consequence, the change in the difference of the logical clocks from t_2 to t_3 is bounded by $\frac{1}{2}n_{k+1}d\frac{4f(1)}{\gamma/4}$. In order to get a change of $n_k\frac{\gamma}{8}$, we set

$$n_{k+1} = \frac{n_k\gamma^2}{64df(1)} \quad .$$

Now, we have two nodes v_i and v_{i+n_k} at t_3 whose logical clocks differ by at least $n_k\frac{\gamma}{8}k + n_k\frac{\gamma}{4} - n_k\frac{\gamma}{8} = (k+1)n_k\frac{\gamma}{8}$. As $n_{k+1} < n_k$, we find another node v_j with $i \leq j \leq i + n_k - n_{k+1}$ such that $L^{j+n_{k+1}}(t_3) - L^j(t_3) \geq (k+1)n_{k+1}\frac{\gamma}{8}$. ■

Thm. 27: *Given any LSA. Then there exists a trace such that two nodes v_i and v_j with $j = i+1$ have at some time t a difference of their local clocks*

$$|L^j(t) - L^i(t)| \geq \frac{\gamma}{8}\frac{\log(n-1)}{\log\left(\frac{8(1+\hat{\rho})}{\hat{\rho}}\log(n-1)\right)} \quad .$$

Proof: The adversary can repeat the construction from Theorem 26 until $n_k = 1$ or $\frac{1}{2}n_kd = 1$, whichever happens first. For the rest of the proof, we assume $d \geq 2$. Initially, the adversary sets $k = 0$ and $n_k = n-1$. Note that for all k, we have

$$n_k = (n-1)\left(\frac{\gamma^2}{64df(1)}\right)^k \quad .$$

For $n_k = 1$, we obtain

$$k = \frac{\log(n-1)}{\log\frac{64df(1)}{\gamma^2}} \quad .$$

At the end of the adversary's construction, we find two neighbor nodes with $n_k = 1$ that have a difference in their logical clocks of $\frac{\gamma}{8}k$. To determine a lower bound on the difference between the logical clocks, we need to solve

$$f(1) = \frac{\gamma}{8}\frac{\log(n-1)}{\log\frac{64df(1)}{\gamma^2}} \quad .$$

The solution is

$$f(1) = \frac{\gamma}{8}\frac{\log(n-1)}{\mathrm{Plog}\left(\frac{8d}{\gamma}\log(n-1)\right)} \quad ,$$

where $\mathrm{Plog}(x)$ is defined as the value z with $x = z \cdot 2^z$. For the values used here, we always have

$$\mathrm{Plog}\left(\frac{8d}{\gamma}\log(n-1)\right) \leq \log\left(\frac{8d}{\gamma}\log(n-1)\right)$$

and hence

$$f(1) \geq \frac{\gamma}{8} \frac{\log(n-1)}{\log\left(\frac{8d}{\gamma}\log(n-1)\right)} \quad .$$

■

4.5 Summary

In this chapter, we examined gradient clock synchronization in a system model with infrequent communication and hence negligible delay uncertainties. We argued that this model reflects typical wireless sensor networks better than the model from [FL04]. We then derived a lower bound for the achievable synchronization quality in our model and discussed its relation to known results.

Further work consists in finding suitable algorithms that attain this bound. We believe that this is not possible with local algorithms. Moreover, the results both from [FL04] and from this chapter could be extended from chains to arbitrary network topologies.

5

Conclusions

We will now summarize the contributions of this thesis and give directions for further work.

5.1 Contributions

In this thesis, we made a number of contributions to the state of the art in the field of time synchronization for mobile ad-hoc networks:

New system model
We presented the zero-delay model as a new system model for the analysis of interval-based time synchronization in mobile ad-hoc networks. We justified our model by observing that clock drift dominates the synchronization error in networks with infrequent communication.

Worst-case bounds
Using our model, we derived worst-case bounds on the quality of interval-based synchronization, namely lower bounds on the synchronization error for a given communication pattern. We showed that the algorithm IM from [MO83] is worst-case-optimal.

Optimal and efficient interval-based synchronization
The algorithm IM is not optimal in the average case. We presented three algorithms, the BP-ISA and Algorithms 3 and 4. These algorithms are also worst-case-optimal, but achieve better synchronization quality in the average case by exploiting the time information that can be gained from the drift diversity of the clocks in the network. We showed that Algorithms 3 and 4 achieve optimal syn-

chronization, albeit at the cost of high memory and communication overhead. We described how limiting the amount of data that is stored and communicated affects the synchronization quality, concluding that the increment in synchronization quality for each additional event stored in the view quickly becomes very small. This means that it typically is not necessary to store and communicate all events to obtain the best possible synchronization.

Mobility
We showed that interval-based synchronization does not need particular communication patterns such as trees or clustered hierarchies. This makes the interval-based approach resilient to node mobility, as there are no broken topologies that have to be repaired. Our simulation results suggest that mobility actually improves interval-based synchronization by increasing the rate of information dissemination through the network.

Gradient clock synchronization in wireless sensor networks
The system model in [FL04] differs considerably from our system model. We showed that a similar lower bound for the error of gradient clock synchronization as in [FL04] exists also in our system model.

5.2 Future Work

A major improvement in the synchronization quality can be achieved by an extension of the clock model: Bounds on the variation $\mathrm{d}\rho(t)/\mathrm{d}t$ of the drift (as proposed in [SS97]) allow to estimate and compensate for the current drift. Back paths can be beneficial also here.

Our simulation results can be extended by varying the parameters. The results presented in this thesis can serve as an indication of where the interesting regions in the parameter space are. For instance, the gain from back paths seems to be largest for those transmission ranges for which all nodes are just barely connected. Another possible extension consists in exploring different mobility models.

For gradient clock synchronization, further work consists in finding suitable algorithms that attain the lower bound or alternatively a better lower bound. Moreover, the results both from [FL04] and from this chapter could be extended from chains to arbitrary network topologies.

Finally, an implementation of the algorithms presented in this thesis on an actual hardware platform would allow to verify the practical benefit of interval-based synchronization.

Bibliography

[AHR96] Hagit Attiya, Amir Herzberg, and Sergio Rajsbaum. Optimal clock synchronization under different delay assumptions. *SIAM Journal on Computing*, 25(2):369–389, 1996.

[BDH+04] Jan Beutel, Matthias Dyer, Martin Hinz, Lennart Meier, and Matthias Ringwald. Poster abstract: Next-generation prototyping of sensor networks. In *Second International Conference on Embedded Networked Sensor Systems (SenSys 2004)*, pages 291–292. ACM, November 2004.

[BDM+04] Jan Beutel, Matthias Dyer, Lennart Meier, Matthias Ringwald, and Lothar Thiele. Next-generation deployment support for sensor networks. Technical Report 207, Computer Engineering and Networks Laboratory, ETH Zurich, November 2004.

[BDMT04] Jan Beutel, Matthias Dyer, Lennart Meier, and Lothar Thiele. Scalable topology control for deployment-support networks. Technical Report 208, Computer Engineering and Networks Laboratory, ETH Zurich, November 2004.

[BDMT05] Jan Beutel, Matthias Dyer, Lennart Meier, and Lothar Thiele. Scalable topology control for deployment-support networks. In *Fourth International Symposium on Information Processing in Sensor Networks (IPSN'05)*, pages 359–363, April 2005.

[Ber00] Jean-Marc Berthaud. Time synchronization over networks using convex closures. *IEEE/ACM Transactions on Networking*, 8(2):265–277, 2000.

[BMT04] Philipp Blum, Lennart Meier, and Lothar Thiele. Improved interval-based clock synchronization in sensor networks. In *Third International Symposium on Information Processing in Sensor Networks (IPSN'04)*, pages 349–358, April 2004.

[BT02] Philipp Blum and Lothar Thiele. Clock synchronization using packet streams. In Dahlia Malkhi, editor, *DISC 2002, Brief Announcements*, pages 1–8, 2002.

[CJBM01] Benjie Chen, Kyle Jamieson, Hari Balakrishnan, and Robert Morris. Span: An energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. In *Mobile Computing and Networking*, pages 85–96, 2001.

[Cri89] Flaviu Cristian. Probabilistic clock synchronization. *Journal of Distributed Computing*, 3:146–158, 1989.

[DBM05] Matthias Dyer, Jan Beutel, and Lennart Meier. Deployment support for wireless sensor networks. In *4. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, ETH Zurich, March 2005.

[DH04] Hui Dai and Richard 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.

[DRSW95] Danny Dolev, Rüdiger Reischuk, Ray Strong, and Ed Wimmers. A decentralized high performance time service architecture. Technical Report 95/26, Institute for Computer Science, University of Lübeck, November 1995.

[EGE02] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[ER02] Jeremy Elson and Kay Römer. Wireless sensor networks: A new regime for time synchronization. In *First Workshop on Hot Topics In Networks (HotNets-I)*, October 2002.

[FL04] Rui Fan and Nancy Lynch. Gradient clock synchronization. In *Twenty-third Annual ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 320–327. ACM Press, 2004.

[Gar79] Floyd M. Gardner. *Phaselock Techniques*. Wiley, 1979.

[GBEE02] Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah Estrin. Locating tiny sensors in time and space: A case study. In *International Conference on Computer Design (ICCD)*, September 2002.

[GKS03] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[GRWE04] Deepak Ganesan, Silvia Ratnasamy, Hanbiao Wang, and Deborah Estrin. Coping with Irregular Spatio-Temporal Sampling in Sensor Networks. *SIGCOMM Computer Communication Review*, 34(1):125–130, 2004.

[HC02] Jason Hill and David Culler. MICA: A wireless platform for deeply embedded networks. *IEEE Micro*, 22(6):12–24, November 2002.

[HS91] Joseph Y. Halpern and Ichiro Suzuki. Clock synchronization and the power of broadcasting. *Distributed Computing*, 5(2):73–82, 1991.

[IEE99] IEEE Computer Society LAN MAN Standards Committee. *IEEE 802.11: Wireless LAN Medium Access Control and Physical Layer Specifications*, August 1999.

[JM96] David B. Johnson and David A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[JOW+02] Philo Juang, Hidekazu Oki, Yong Wang, Margaret Martonosi, Li-Shiuan Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *ASPLOS X*, San Jose, USA, October 2002.

[KM04] Branislav Kusy and Miklós Maróti. Flooding time synchronization in wireless sensor networks. Unpublished, 2004.

[Lam78] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[Lis91] Barbara Liskov. Practical uses of synchronized clocks in distributed systems. In *10th Annual ACM Symposium on Principles of Distributed Computing (PODC'91)*, pages 1–10, August 1991.

[LL84] Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2/3):190–204, August/September 1984.

[LR04] Qun Li and Daniela Rus. Global clock synchronization in sensor networks. In *IEEE InfoCom*, 2004.

[MBT04] Lennart Meier, Philipp Blum, and Lothar Thiele. Internal synchronization of drift-constraint clocks in ad-hoc sensor networks. In *Fifth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2004)*, pages 90–97, May 2004.

[MBT05] Lennart Meier, Philipp Blum, and Lothar Thiele. Interval-based clock synchronization is resilient to mobility. In *Second IEEE International Conference on Mobile Ad Hoc and Sensor Systems (MASS 2005)*, November 2005.

[Mei05] Lennart Meier. Interval-based clock synchronization for ad-hoc sensor networks. In *4. GI/ITG KuVS Fachgespräch Drahtlose Sensornetze*, ETH Zurich, March 2005.

[MFNT00] Michael Mock, Reiner Frings, Edgar Nett, and Spiro Trikaliotis. Clock synchronization in wireless local area networks. In *12th Euromicro Conference on Real Time Systems*, pages 183–189, June 2000.

[MFT05] Lennart Meier, Pieric Ferrari, and Lothar Thiele. Energy-efficient bluetooth networks. Technical Report 204, Computer Engineering and Networks Laboratory, ETH Zurich, January 2005.

[Mil] David L. Mills. Bibliography on computer network time synchronization. Available on-line at http://www.eecis.udel.edu/˜mills/biblio.html.

[Mil91] David L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[Mil95] David L. Mills. Improved algorithms for synchronizing computer network clocks. *IEEE/ACM Transactions on Networks*, 3(3):245–254, June 1995.

[MKSL04] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, pages 39–49, New York, NY, USA, November 2004. ACM Press.

[MO83] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. In *Second Annual ACM Symposium on Principles of Distributed Computing (PODC'83)*, pages 295–305. ACM Press, 1983.

[MR03] Sayan Mitra and Jesse Rabek. Power efficient clustering for clock synchronization in dynamic multi-hop networks. Unpublished, 2003.

[MT05a] Lennart Meier and Lothar Thiele. Brief announcement: Gradient clock synchronization in sensor networks. In *Twenty-Fourth Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2005)*, page 238, July 2005.

[MT05b] Lennart Meier and Lothar Thiele. Gradient clock synchronization in sensor networks. Technical Report 219, Computer Engineering and Networks Laboratory, ETH Zurich, April 2005.

[Nor00] Raffaele Noro. *Synchronization over Packet-Switched Networks: Theory and Applications*. PhD thesis, EPFL, Lausanne, Switzerland, 2000.

[PSR94] Boaz Patt-Shamir and Sergio Rajsbaum. A theory of clock synchronization. In *26th Annual ACM Symposium on Theory of Computing*, pages 810–819, May 1994.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterli, and Brian P. Flannery. *Numerical Recipes in C, 2nd Edition*. Cambridge University Press, 1992.

[RBM05] Kay Römer, Philipp Blum, and Lennart Meier. Time synchronization and calibration in wireless sensor networks. In *Handbook of Sensor Networks: Algorithms and Architectures*. John Wiley & Sons, September 2005.

[Röm01] Kay Römer. Time synchronization in ad hoc networks. In *2nd ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2001)*, pages 173–182. ACM Press, October 2001.

[Röm03] Kay Römer. Temporal message ordering in wireless sensor networks. In *IFIP Mediterranean Workshop on Ad-Hoc Networks*, pages 131–142, June 2003.

[SA05] Weilian Su and Ian F. Akyildiz. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 13(2):384–397, 2005.

[SS97] Ulrich Schmid and Klaus Schossmaier. Interval-based clock synchronization. *Real-Time Systems*, 12(2):173–228, 1997.

[SV03] Mihail L. Sichitiu and Chanchai Veerarittiphan. Simple, accurate time synchronization for wireless sensor networks. In *IEEE Wireless Communications and Networking Conference (WCNC'03)*, March 2003.

[SY04] Fikret Sivrikaya and Bülent Yener. Time synchronization in sensor networks: A survey. *IEEE Network*, 18(4):45–50, July 2004.

[vGR03] Jana van Greunen and Jan Rabaey. Lightweight time synchronization for sensor networks. In *2nd ACM International Workshop on Wireless Sensor Networks and Applications*, pages 11–19, September 2003.

[VRC97] Paulo Verissimo, Luis Rodrigues, and Antonio Casimiro. Cesiumspray: a precise and accurate global time service for large-scale systems. *Real-Time Systems*, 3(12):243–294, 1997.

[WESW98] Hagen Woesner, Jean-Pierre Ebert, Morten Schlager, and Adam Wolisz. Power saving mechanisms in emerging standards for wireless LANs: The MAC level perspective. *IEEE Personal Communications*, 5(3):40–48, June 1998.

[YHE02] Wei Ye, John Heidemann, and Deborah Estrin. An energy-efficient MAC protocol for wireless sensor networks. In *IEEE INFOCOM*, June 2002.

[ZLX02] Li Zhang, Zhen Liu, and Cathy Honghui Xia. Clock synchronization algorithms for network measurements. In *IEEE INFOCOM*, June 2002.

Curriculum Vitae

Personal data:

Full name:	Lennart Lauri Rudolf MEIER
Date of birth:	1976-03-15
Place of birth:	Heidelberg (Germany)
Citizenship:	Finnish and German
Mail address:	Albisriederstrasse 84, 8003 Zurich, Switzerland
E-mail:	lennart.meier@postmail.ch

Education:

1982 – 1995	schooling in Helsinki, Berlin, and Rome
1995-06	school-leaving examination (“Abitur”)
1995-07 – 1996-06	military service in the Finnish Defence Forces
1996-10	start of computer science studies at Saarland University, Saarbrücken (Germany)
1998-06	scholarship of the German National Academic Foundation
2000-06	diploma in computer science
2000-08 – 2002-01	Ph.D. student with Prof. Dr. Ueli Maurer, ETH Zurich research area: cryptography and information security
2002-02 – 2005-12	Ph.D. student with Prof. Dr. Lothar Thiele, ETH Zurich research area: synchronization in mobile ad-hoc networks
2005-12	completion of Ph.D. studies

Professional experience:

2000-08 – 2002-01	assistant with Prof. Dr. Ueli Maurer, ETH Zurich
2002-02 – 2005-12	assistant with Prof. Dr. Lothar Thiele, ETH Zurich