Diss. ETH No. XXXXX

Guaranteed Time Synchronization in Wireless and Ad-Hoc Networks

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZÜRICH

for the degree of
Doctor of Sciences

presented by
PHILIPP BLUM
Ing. dipl. en Microtechnique, EPF Lausanne
born March 2, 1974
citizen of Beromünster, LU

accepted on the recommendation of
Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Wolfgang Fichtner, co-examiner

2004

Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory

TIK-SCHRIFTENREIHE NR. XX

Philipp Blum

Guaranteed Time Synchronization in Wireless and Ad-Hoc Networks

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Diss. ETH No. XXXXX

Prof. Dr. Lothar Thiele, examiner
Prof. Dr. Wolfgang Fichtner, co-examiner

Examination date: XX XX, 2004

Abstract

Time synchronization is an important service in many distributed systems. The goal of time synchronization is to provide the nodes of such a system with a common, i.e. synchronized, time base. Synchronized time allows for coordinated actuation and sensor fusion. Time synchronization requires that the nodes have local clocks and can then be achieved by communication among the nodes. The reasons why time synchronization is never perfect are the variable delay of messages exchanged by the nodes and the variable speed of the nodes' local clocks, i.e. clock drift. Clock-synchronization algorithms aim at providing the nodes with a means to translate their local time to synchronized time.

This thesis is about guarantees on the quality of time synchronization that can be provided for real systems. Such guarantees can be obtained by measuring the synchronization quality in a real system or by using analytical methods and formal system models. In the first part of this thesis, we apply both approaches in the context of the wireless-loudspeakers application. This application poses particularly hard requirements on the synchronization quality since the human ear can discern a temporal misalignment in the order of a few microseconds between correlated audio channels. The main contributions are listed in the following.

- A novel class of clock synchronization algorithms is introduced, which, in contrast to known state-of-the-art algorithms, achieve sufficient synchronization quality in a wide range of conditions, that is in various operation modes of the wireless network and in situations of heavy network traffic.

- A novel framework for the optimization and comparison of clock-synchronization algorithms is presented. The framework combines trace-based simulation and evolutionary optimization to provide for a realistic and reproducible evaluation and a fair comparison.

- The comparison of empirical and analytical worst-case results allows to argue about the suitability of system models. A novel delay model is proposed that captures the relevant properties of measured delay sequences more accurately than previously known delay models.

In the second part, this thesis also contributes to the understanding of the worst-case synchronization quality that can be achieved in large-scale systems. Here the goal is to design the time-synchronization service so that a high synchronization quality can be achieved with minimal overhead in terms of communication among the nodes. In large-scale systems, some nodes and links between nodes may fail, and nodes may be mobile. A novel class of algorithms is proposed that has the advantage over known approaches that it is completely local. No global configuration is required and thus a high resilience against node and link failures is achieved.

Finally, it is shown how this approach for organizing time synchronization in large-scale networks can be combined with the algorithms for accurate point-to-point synchronization presented in the first part of the thesis.

Kurzfassung

Zeitsynchronisation ist ein wichtiger Dienst in vielen verteilten Systemen. Das Ziel ist es, den Knoten in einem solchen System eine gemeinsame, also synchronisierte, Zeitbasis zur Verfügung zu stellen. Damit werden koordinierte Aktionen und die gemeinsame Auswertung von Beobachtungen ermöglicht. Ein Zeitsynchronisationsdienst basiert darauf, dass alle Knoten eine lokale Uhr besitzen und miteinander kommunizieren können. Perfekte Synchronisation kann nicht erreicht werden, weil die Laufzeiten von Nachrichten und die Laufgeschwindigkeiten der lokalen Uhren variabel sind. Uhrensynchronisationsalgorithmen erlauben es den Knoten, die lokale Zeit in synchronisierte Zeit umzurechnen.

Die vorliegende Arbeit beschäftigt sich mit Garantien zur Synchronisationsqualität, welche in realen Systemen erreicht werden kann. Solche Garantien können durch Messungen oder mittels analytischer Methoden und formaler Modelle gefunden werden. Im ersten Teil dieser Arbeit werden beide Methoden im Kontext von drahtlosen Lautsprechern angewendet. Diese Anwendung stellt besonders harte Anforderungen an die Synchronisationsqualität, da das menschliche Ohr zeitliche Verschiebungen von wenigen Mikrosekunden zwischen zusammengehörigen Audiokanälen wahrnehmen kann. Nachfolgend die wichtigsten Resultate:

- Eine neue Klasse von Uhrensynchronisationsalgorithmen wird vorgestellt, mit welchen im Gegensatz zu bekannten Algorithmen unter verschiedenen Randbedingungen und auch unter starker Netzwerkbelastung eine für drahtlose Lautsprecher genügende Synchronisationsqualität erreicht wird.

- Eine Methode zur Optimierung und zum Vergleich von Uhrensynchronisationsalgorithmen wird präsentiert. Die Methode kombiniert Trace-basierte Simulation mit evolutionärer Optimierung, was realistische und reproduzierbare Resultate sowie einen fairen Vergleich von verschiedenen Algorithmen erlaubt.

- Die Gegenüberstellung von empirischen und analytischen Resultaten erlaubt es, die Eignung verschiedener Systemmodelle kritisch zu hinterfragen. Ein neues Modell für Nachrichtenlaufzeiten wird vorgeschlagen, welches die relevanten Eigenschaften von Laufzeitsequenzen besser beschreibt als bekannte Modelle.

In einem zweiten Teil enthält die vorliegende Arbeit Beiträge zum Verständnis der erreichbaren Synchronisationsqualität in Systemen mit sehr vielen Knoten. Hier ist es das Ziel, den Sychronisationsdienst so zu gestalten, dass eine gute Qualität mit minimalen Aufwand bezüglich der Zahl benötigter Nachrichten erreicht wird. Dabei muss berücksichtigt werden, dass in grossen Systemen Knoten ausfallen und/oder mobil sein können. Eine neue Klasse von Algorithmen wird vorgestellt, welche den Vorteil haben, dass sie rein lokal, also ohne globale Konfiguration arbeiten und daher robust gegen Ausfall und Mobilität von Knoten sind.

Abschliessend wird gezeigt, wie dieser Ansatz zur Organisation der Synchronisation in grossen Netzen mit den im ersten Teil der Arbeit eingeführten Punkt-zu-Punkt-Synchronisationsalgorithmen kombiniert werden kann.

I would like to thank

- Prof. Dr. Lothar Thiele for the valuable discussions concerning this research,
- Prof. Dr. Wolfgang Fichtner for his willingness to be the co-examiner of my thesis,
- Georg Dickmann from BridgeCo and Men Muheim for many interesting discussions about synchronization for audio applications, and
- Lennart Meier for the profitable cooperation concerning synchronization in ad-hoc networks.

Contents

1

Introduction

The quality of time synchronization in computer networks not only depends on the synchronization algorithm that is used, but also on non-deterministic system stimuli. Therefore, the synchronization quality of a particular system is variable. This thesis is about guarantees on the quality of time synchronization, that is about determining how bad the synchronization quality can be in the worst case.

There are basically two ways to determine such guarantees on the synchronization quality, which are illustrated in Fig. 1. (i) The synchronization quality can be measured. The problem is that no matter how many measurements are performed, one cannot be sure that the worst case has been covered. Measurements deliver an optimistic estimation of the worst-case synchronization quality. (ii) The system under consideration can be abstracted into an analytical model. In this model, rigorous proofs of the worst-case synchronization quality are possible. The problem here is to make the right abstractions, since derived guarantees about the worst case are valid for the model and not for the actual system. If all assumptions of a model are true, then the derived guarantee must be pessimistic. If these assumptions are wrong, then the guarantee may be pessimistic or optimistic.

In this thesis, we will use empirical and analytical methods to approximate the worst-case synchronization quality in two different kinds of networks, that is (i) in small wireless networks used for audio distribution and (ii) in large-scale wireless sensor networks. The empirically obtained results will be used to discuss the suitability of a variety of analytical models. As illustrated in Fig. 1, the measured worst case does not always allow to differentiate between a wrong and a correct but incomplete model. Therefore, special emphasis is put on quantifying the analytical results for concrete systems, which allows us to

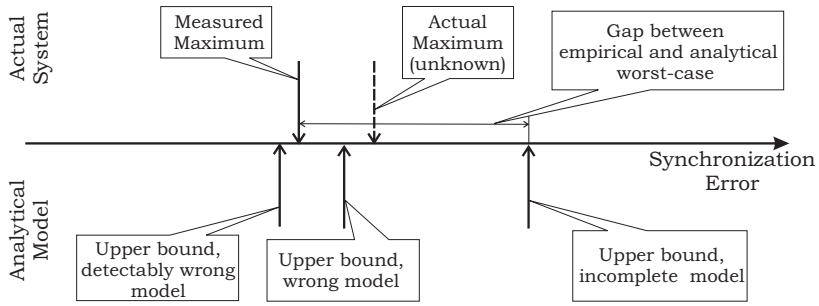


Fig. 1: Empirical and analytical methods can be used to approximate the actual maximal synchronization error (worst case). Neither of these methods provides guarantees on the actual system in a strict sense.

argue about a particular model in terms of the gap between the measured and the analytically derived worst-case synchronization quality.

In Section 1.1, a short overview of the fundamental problems involved in time synchronization in computer networks is presented. In Section 1.2, we describe several application domains that require tight synchronization. A short overview of known results and a survey of related research topics is given in Section 1.3. An outline of this thesis is provided in Section 1.4.

1.1 Clock Synchronization in a Nutshell

We start our discussion by giving three examples explaining what clock synchronization is about.

Ex. 1: **(Clock synchronization is difficult to establish.)** *Let us assume that we have two clocks A and B that run at exactly the same speed, and we want to adjust the offset of clock B, so that B shows the same time as A. This is no problem if the offset of B can be adjusted while reading the time from A. But sometimes this is not possible, e.g. because A is mounted on the kitchen wall, while B is the internal clock of the car radio. Now, setting the offset of B is more difficult: First we read the time from A, then we go to B and adjust B according to the reading of A. But the clocks are not yet synchronized, since A did not stop during the time we walked to B. The obvious solution to this problem is to add the time it took us to walk from A to B to the reading of A when adjusting B. In other words, the timestamp t_1 contained in the message from A to B has to be corrected by the delay d of this message. If this delay is known, then the problem is solved. But as the clock B is not a priori synchronized with clock A, it is not possible to measure the delay using clocks A and B. Thus, we face the typical chicken and egg problem, or put mathematically: Given t_1 and the relation $t_2 = t_1 + d$, what are the values of t_2 and d?*

The example showed that unknown message delay is an obstacle to achieving clock synchronization. This problem is very old[1] and many smart solutions have been found for special instances. Most notably the *round-trip experiment* has proven to be useful in many situations: First, we read the time t_1 from clock A and send a message to B. When the message arrives, we read t_2 from clock B and send a message back to A, where t_3 is read. If it can be assumed that the delay from A to B is equal to the delay from B to A, we know that the time of A was $\frac{1}{2}(t_1 + t_3)$ when the time of clock B was t_2, regardless of the actual delay the messages encountered.

Recent advances in computer-network technologies have once again made clock synchronization an active area of research. Packet-switching technologies in the WAN and statistical multiplexing in the LAN allow more economic use of bandwidth than circuit switching and fixed TDMA schedules, and thus are rapidly expanding to new application domains. The downside of these technologies concerning clock synchronization is that the delay of a packet[2] is less predictable. The delay is strongly influenced by the current network load and the assumption of symmetric delays between any two network nodes becomes unrealistic.

Small differences in the rate of clocks are another obstacle to clock synchronization, as is shown in the next example.

Ex. 2: **(Clock synchronization is difficult to maintain.)** *Let us assume that we have two clocks A and B that run at slightly different speeds. Let us further assume that at time t_1 clock B has been synchronized to clock A and both clocks show exactly the same time. Obviously, at time $t_2 > t_1$ this is not the case anymore. Therefore, synchronization has to be re-established periodically.*

A longer re-synchronization period can be chosen if the rate of B can be adjusted so that it is close to the rate of A. Adjusting the rate requires that the rate difference is known. Determining the rate difference between two clocks is thus an important aspect of clock synchronization. It is feasible if (i) the rate difference is constant and (ii) the offset between clocks A and B is known at two times t_1 and t_2. Both requirements are never satisfied in a real system and thus various approximation techniques have been developed.

A third problem arises when a large number of clocks has to be synchronized to a common time. Assume that synchronization can only be performed pairwise, that is you can read the time of two clocks simultaneously and then adjust either one of them or both.

Ex. 3: **(Clock synchronization is difficult to organize.)** *If I told you that the time when writing these lines is 9 a.m. according to my wristwatch and 3 p.m. by my*

[1] See e.g. the informal discussion of clock synchronization for the European railroad system in the early 20th century in [Gal03].

[2] The terms packet and message are used as synonyms in this thesis, as all messages used in the algorithms presented in later sections are so short that they can be transported in a single packet.

desktop computer's time, could you tell which information is better? You could not, unless additional information about these two sources of time information is available. For example if you know that the desktop computer regularly synchronizes with a stratum-one time server using NTP while the wristwatch is an old mechanical one, which I forgot to wind up this morning, then it is quite clear that the time actually is about 3 p.m.

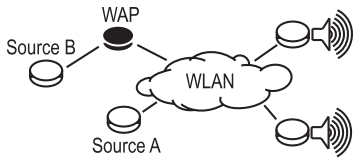
This example shows that the organization of clock synchronization in large-scale networks requires some mechanisms that defines how the synchronized time is "routed". For example, the clocks that have to be synchronized can be organized in a tree and every child node can then synchronize to its parent node. But such a solution has the drawback that the construction of this tree structure does not come for free and may be difficult to achieve if the network topology is constantly changing due to node mobility or node failures.

1.2 Applications of Synchronized Clocks

In this section, we sketch four application domains in which synchronized clocks play a central role. A description of the application is followed by a short discussion of the requirements on the synchronized clocks. The requirements are different in all four cases and they are summarized in Table 1. A more detailed discussion of requirements and metrics for the synchronization quality is given in Chapter 2. The case of media distribution is studied in detail in Chapter 3. The case of wireless sensor networks is studied in Chapter 4. The cases of circuit emulation and distributed control have been selected because we believe that the methods and algorithms presented in this thesis are also applicable in these contexts and promise substantial benefits over existing solutions.

1.2.1 Media Distribution

In recent years, many new consumer-electronics (CE) technologies have entered our homes. As these devices start to integrate networking capabilities in order to provide new and improved services, wire-bound connections more and more become a burden. Wireless technologies like the IEEE 802.11 standards seem to be a promising alternative for certain CE applications. Wireless loudspeakers are a particularly interesting application, since current home-cinema systems require up to 8 loudspeakers distributed in the living room. At the same time, this application poses hard real-time challenges. A temporal offset between correlated audio channels causes different psychoacoustic effects, as illustrated in Fig. 2. Such an offset results from oscillator drift in the loudspeakers and from the variability of packet delays. Proper synchronization of the loudspeakers and of the audio source is thus essential. The requirements on the synchronization quality are (i) keeping the offset constantly below 1 millisecond, (ii) keeping offset variations below 100 microseconds (a constant pan is no problem, as the



Channel Offset	*Psychoacoustic Effects*
$> 10\mathrm{ms}$	Source separation
$> 1\mathrm{ms}$	Timbre
$> 100\mu\mathrm{s}$	Pan
$> 10\mu\mathrm{s}/10\mathrm{s}$	Noise

Fig. 2: Wireless-Loudspeakers Application: Two or more loudspeakers are connected over an IEEE 802.11b network to a source of audio streams. The source is connected to the wireless network either directly or via a wireless access point (WAP). On the right, the time scales of different psychoacoustic effects are summarized: If the offset is larger than $10\mathrm{ms}$, the two channels are perceived as separate sources. Around $1\mathrm{ms}$, the sound's characteristics are modified. Starting at $100\mu\mathrm{s}$, the offset can also change the directional perception of the source. Even smaller offsets can cause noise if the offset is rapidly changing (by more than $10\mu\mathrm{s}$ in an interval of less than $10\mathrm{s}$ length).

balance between the channels has to be adjusted according to the user's position anyway), and (iii) avoiding fast offset variations above 10 microseconds. Additionally, these criteria have to be achieved within a short setup time, as a user is not willing to wait more than a few seconds for the system to become operational.

1.2.2 Ad-hoc Networks

Ad-hoc networks have lately received a lot of interest from academia, e.g. in the form of wireless sensor networks [ADL+98], [ASSC02], or more generally mobile information and communication services that are based on self-organization [VG01].

Clock synchronization is an important service in these networks. For example, the fusion of distributed sensor data may require knowledge about the chronology of the sensor observations [Röm01, Röm03a]. The energy consumption of wireless devices can be reduced by synchronous power-on and shutdown of the wireless-communication circuits of a sender – receiver pair [CJBM01, WESW98]. Synchronization is required for federating multiple sensor nodes into an array with augmented sensing capabilities, e.g. for the localization of audio sources as proposed in [GBEE02] and [SBM+04].

Though the requirements on synchronization are substantially different for the various applications envisaged for ad-hoc networks, there are some common challenges that are different from those in infrastructure-based networks [ER02, Röm03b]. (i) *Energy efficiency:* Synchronization can only be achieved and maintained by communication, which is expensive in terms of energy. As sensor nodes are autonomous, battery-powered devices, energy-awareness is a key requirement for every service on such a node, including clock synchronization. (ii) *Ad-hoc deployment:* The clock-synchronization service must not rely on any a-priori configuration or infrastructure. (iii) *Robustness under node mobility and node failure:* There is no guarantee of stable connections between nodes.

(iv) The aforementioned acoustic-ranging applications also demand synchronization accuracy in the microsecond range, while temporal ordering of events often requires only millisecond accuracy.

1.2.3 Circuit Emulation

Recent developments in transmission and switching technologies create new opportunities to converge telephony and data services. Given the emerging deployment of switched Gigabit-Ethernet in MANs, these networks can be employed to emulate telephony circuits to offer connectivity for legacy telephony systems such as GSM base stations and PBXs[3]. Recent [Nor00] and ongoing research projects [Fie04] propose that so-called circuit emulation adapters provide a T1/E1[4] carrier to which the legacy components are connected.

In order to prevent buffer under- or overflow in the circuit-emulation adapters, the clock *rates* of these devices need to be synchronized across the Gigabit-Ethernet. As such systems are continuously in service, the setup time is not of concern. Energy and network bandwidth are available in largely sufficient quantities.

1.2.4 Distributed Control

Recently, the Time-Triggered Architecture (TTA) has been proposed [MBSP02] as a distributed computing platform for large, complex and safety-relevant systems in the aerospace and automotive domain. It is argued that TTA facilitates the composition of components into a complex system. A system using TTA is based on globally coordinated, static schedules of all activities and thus requires synchronized clocks in all components of the system. Clock synchronization is achieved with microsecond accuracy using the Time-Triggered Protocol (TTP) [KG94], which is in essence a TDMA protocol. Similar levels of accuracy are also achieved by the competing protocols FlexRay [FC] and Time-Triggered CAN [Gmb, HMFH02]. Also a short setup time is necessary.

Major car manufacturers lately seem to prefer FlexRay over TTP [Mur03], the main argument being that FlexRay additionally reserves time slots for event-triggered communication, and thus provides more flexibility. Of course, the use of standard CDMA communication technologies would provide even more flexibility and presumably better economies of scale[5]. In this thesis, we show that similar levels of clock-synchronization accuracy can be achieved in standard Ethernet networks. It seems possible that the vast amount of bandwidth already available in today's wired and even wireless Ethernet technologies may allow reliability and safety problems to be partially circumvented in the future.

[3] Short for private branch exchange, a private telephone network used within an enterprise. Users of the PBX share a certain number of outside lines for making telephone calls external to the PBX. From http://www.webopedia.com

[4] Standard TDMA carrier used in telecommunications. See e.g. [Tan96].

[5] In the automation industry, where safety concerns are less acute, this trend is already becoming a reality as reflected by the new IEEE 1588 standard [Edi02]. A comprehensive

	Media Distribution	*Ad-Hoc Sensor Networks*	*Circuit Emulation*	*Distributed Control*
Offset	•	•(•)		••
Rate	••		••	••
Setup Time	••			•
Energy		••		
Fault Tolerance		••	•	••

Tab. 1: Comparison of different applications' requirements on clock synchronization. Two dots indicate mission-critical issues, one dot stands for issues with some importance.

1.3 Research Topics in Clock Synchronization

The problem of clock synchronization has many aspects that are active research topics. In this section, we give an overview of the most important areas, focusing on real-time clock synchronization, implemented as software processes in a distributed system. We do not cover hardware clock synchronization[6], but discuss hardware assistance for high-performance software clock synchronization. Also, logical (vector) time is not considered[7]. Our discussion starts with theoretical aspects, namely contributions on lower bounds on the achievable accuracy[8]. Later, contributions in selected application areas with specific requirements on clock synchronization are discussed.

1.3.1 Optimal Synchronization

The achievable accuracy depends on the system model, consisting of (i) the network model, (ii) bounds on message delays and (iii) assumptions on clock drift. A lower bound for completely connected systems with bounded delay and without drift is given in [LL84], an algorithm achieving the lower bound is also presented. [HMM85] extends the lower bound to arbitrary networks. [BW01] presents closed-form bounds for chains, meshes and hypercubes. The authors of [LL84], [HMM85], and [BW01] give *worst-case-optimal* bounds, viewing the system as an adversary that intentionally chooses message delays and clock drifts in such a way as to make synchronization as difficult as possible.

[AHR93], [PSR94], and [OPS99] use the stronger criterion of *optimality in every execution.* [AHR93] presents lower bounds assuming either (i) known lower and upper delay bounds, (ii) a known bound on round-trip bias or (iii)

overview of existing solutions in this domain is given by [Sch03].

[6]For an introduction to phase-lock techniques see [Gar79]. An good introduction to GPS is [HWLC97].

[7]Logical time is only concerned with determining precedence relations [Lam78] of events, but not with the real time when events actually occur. For an introduction, see [Mor85] and [Mat93].

[8]As the terms *accuracy*, *precision*, etc. are not used consistently in the literature, we use *accuracy* here to represent the quality of synchronization in a generic way. A formal definition of the terms as used in this thesis is given in Chapter 2.

known lower and upper bounds on receive-time difference. In [PSR94], these results are extended to the on-line case, and [OPS99] incorporates drifting clocks. A quite different approach to optimality is presented in [KEES03] and similarly in [HS03], where optimality in a statistical, maximum-likelihood sense is proposed.

In some situations, it is desirable that synchronized clocks not only fulfill a given accuracy requirement, but that the nodes *themselves know* the actual accuracy. [MO83, Mar84] and [SS97] present the idea of interval-based synchronization, providing algorithms that compute an interval in which the reference time is contained. The algorithms presented in [PSR94, OPS99] compute an error margin, which is essentially equivalent. [Sch97, Sch98] extend the interval-based approach to rate synchronization. Also the convex-closure approach of [Ber00] falls into the category of interval-based synchronization.

[SC90] shows that optimal accuracy can be achieved with continuous amortization instead of discrete clock adjustments; similar ideas are found in the patent [Str96].

1.3.2 Accurate Synchronization

Often, the target accuracy of an application is better than the deterministically optimal synchronization accuracy [LL84, HMM85, BW01]. Three back doors have been found, *probabilistic* synchronization, *a-posteriori* synchronization and *hardware-assisted* synchronization[9]. The notion of probabilistic synchronization stands for trade-off strategies between accuracy and the probability of achieving this accuracy[10]. The probabilistic algorithm of [Cri89] uses a dynamically determined number of round-trip messages to achieve a desired accuracy. A better accuracy can be achieved by allowing more round-trip messages. A more static, a-priori approach is taken in [Arv94]: Several unidirectional messages are used to perform a linear-regression analysis of the clock offset and drift. Assuming normal distribution of message delays, the probability of achieving a given accuracy with a given number of messages can be computed. Good performance is achieved if the message delay approximately follows a Gaussian distribution[11]. Linear regression can also be interpreted as a system-identification technique [LGX00]. Linear regression has been found to converge faster than e.g. phase-locked-loop algorithms [Nor00].

The *a-posteriori agreement* technique has been proposed in [HS91], [VR92] and [VCR93]. It makes use of the observation that the receive times of a broad-

[9] The latter two approaches are not actually back doors, they rather allow system constants, as for example message-delay variability, to be decreased but are still in line with the results from [LL84, HMM85, BW01].

[10] In contrast to the common use of the notion *probabilistic* in algorithm theory, probabilistic synchronization does not mean algorithms that make random decisions but algorithms that have a certain probability of achieving a given accuracy.

[11] This is the case e.g. for the RF communication used on the Berkeley Motes, which is the reason why linear regression has been successfully applied on this platform [EGE02, HS03, MKSL04a].

cast message are often more closely aligned and less variable than the send and receive times of a broadcast or unicast message. Implementations of the technique are presented in [VRC97] and for an IEEE 802.11 wireless network in [MFNT00]. Also the RBS algorithm [EGE02] is based on this technique. The a-posteriori agreement can be applied easily e.g. in a CAN network where the sender of broadcast messages itself receives the message and can time-stamp this reception. This time stamp can then be appended to the next broadcast message, thus time-stamp distribution does not cost additional messages. In most networks, a sender of broadcast messages does not itself receive the message. Then the a-posteriori agreement technique becomes more complicated, as the receive times have to be collected and distributed.

To achieve an accuracy in the microsecond range, the use of special *hardware assistance* has been proposed. [HC02, GKS03, EGE02, vGR03, MKSL04b] use low-level access to the Berkeley Motes' RF interface, which has a particularly small delay variability. [LMC99] presents an implementation of the probabilistic algorithm of [Cri89] for the Myrinet technology, achieving low microsecond accuracy. [MDG01] discusses hardware enhancements of standard PCs for time-stamping received and sent packets. [MFNT00] presents a driver-level implementation of an a-posteriori algorithm achieving 150 microseconds on a 802.11b network. [Mil93] discusses hardware and software interfaces to connect a PC to a source of reference time signals (e.g. Pulse Per Second, DCF, GPS, etc.).

1.3.3 Synchronization in Large Systems

In large systems, the message complexity of many clock-synchronization algorithms becomes intolerable. Several solutions have been proposed to solve the problem. The widely deployed NTP [Mil91] is based on a hierarchy of time servers to reduce message complexity. [VRC97] presents a similar scheme. Another approach is to use unidirectional communication of timing information only, which provides perfect scalability concerning the number of nodes that listen to a source of timing information. [Arv94] presents a statistical analysis of achievable accuracy for such a unidirectional scheme. In [DRS94], the theoretical foundations of *eavesdropping* synchronization are given, followed up by the more practically-oriented [DRSW95]. This concept allows the advantages of unidirectional synchronization and the guaranteed-interval approach to be combined, but relies on complicated assumptions about the behavior of the communication link. The eavesdropping technique is patented [SW93].

Even unidirectional schemes can run into scalability problems. [HL02] reports occasional failures of the Time Synchronization Function (TSF) built into the IEEE 802.11 standard. This synchronization scheme is essentially equivalent to an algorithm presented in [Lam78]: All clocks constantly broadcast their local time. Whenever a time stamp is received that is ahead of the local time, the local time is updated accordingly. All clocks will thus follow the fastest clock in the system. The problem of this scheme occurs when the node with the

fastest clock cannot broadcast its time due to contention on the communication medium.

1.3.4 Fault-Tolerant Synchronization

In large distributed systems, it is unavoidable that some components eventually fail. Research on fault-tolerant synchronization covers questions of how many faults of nodes and links can be tolerated. Node faults include node crashes and Byzantine behaviors. Link faults include permanent failure, message omission and late or early delivery (violating a-priori bounds). [SLWL90] presents an introduction to various system and fault models and discusses important algorithms. It is shown that in an arbitrary network, f link faults can be tolerated if the connectivity is at least $2f$. If an authentication service is available[12], then only $f + 1$ connectivity is required. In a completely connected network, authentication is of no use. Regarding node faults, it is shown that synchronization is possible if at least $3f$ nodes participate. If authentication is available, only $2f$ nodes are required. Later work [DHSS95] treats the question of processor joins. It is found that joins are possible if at least half the nodes are not faulty and authentication is possible. [dAB94, FC95] introduce further variants of convergence functions. [BHHN00] integrates previous results in fault-tolerance with a drifting clock model. The consequence is that synchronization is a never-ending process, and that therefore the number of faults cannot reasonably be bounded. [SW99] extends the interval-based algorithms of [Sch97, Sch98] with fault-tolerance mechanisms. A comparison of many fault-tolerant clock-synchronization algorithms is given in [AP98].

1.3.5 Ad-hoc and Energy-Efficient Synchronization

Similarly to other application areas, the contributions on clock synchronization cover single-hop and multihop issues. The Berkeley Motes platform is used by many researchers for an implementation of their ideas, which allows the comparison of their results. For single-hop synchronization, it is disputed whether the a-posteriori agreement technique, as used by the RBS algorithm [EGE02], achieves better results than round-trip techniques [GKS03, GKAS03, SV03, vGR03] or unidirectional [MKSL04b] communication of time information. Reported accuracies of all these algorithms lie in the low microsecond range.

multihop synchronization schemes organize the distribution of time information in a large network. [MR03] proposes a clustering scheme for RBS that is aware of energy restrictions. The TPSN algorithm [GKS03] first creates a static synchronization hierarchy before time information is distributed from a single reference node at the top of the hierarchy. The FTSP algorithm [MKSL04b] provides a more dynamic scheme that can tolerate node failures and node mobility.

[12] Node A receives a message from node B via node C. Authentication allows node A to verify if the contents of the message are exactly those produced by node B, regardless of the behavior of the intermediate node C.

Another focus of research on clock synchronization in the area of wireless sensor networks is energy efficiency. The concept of *post-facto* synchronization has been proposed by [EE01] and [ER02] to reduce energy consumption: In contrast to *always-on* synchronization, a post-facto approach only synchronizes the time of nodes that actually *have* collected information that needs to be time-stamped, thus synchronization happens only *on demand*[13]. [Röm01] gives an algorithm that implements post-facto synchronization. An interesting feature is that instead of time, time intervals are measured and converted from one time scale to another strictly on an as-needed basis. The deficiency of the approach is that the scheme is not applicable to coordinated actuation.

1.3.6 Network-Delay Measurements

An active research area in its own right is network-delay measurement. The basic question here is the following: Given unsynchronized local send and receive time stamps of a sequence of one-way packets (most often in a WAN context), is it possible to determine and compensate relative clock drift[14] and discontinuities in one of the clocks? Important publications in the field are [Pax97, Pax98, MST99] presenting new algorithms, [ZLX02] giving a comparison of several known algorithms and [PV01], [AGR03] discussing measurements systems.

Though it is clear that this question comes very close to the clock-synchronization problem, neither of the communities cites the work of the other very much. One big difference in the problem statements that may be responsible for this is that the network-delay problem is approached off-line, whereas clock synchronization (at least in the case of drifting clocks) is an on-line problem.

1.3.7 Audio

Synchronization of media streams most often refers to implicit synchronization of one or more streams using buffer fill levels as input rather than time stamps. The goal of such algorithms is to avoid buffer over- and underflows and to achieve lip-synchronization between audio and video streams. An overview of this kind of synchronization algorithms is given in [IT00] and [LS02]. Formal definitions of synchronization properties for stream-synchronization are given in [Ste90].

Real-time synchronization with an accuracy in the millisecond range is required for *interactive-performance* systems. Such systems allow geographically distant artistic performers to cooperate, based on communication of MIDI events. The clock-synchronization aspects of such systems are discussed e.g. in

[13] We believe that the dichotomy always-on vs. post-facto is slightly misleading and suggest always-on vs. on-demand instead. The post-facto principle only applies to sensing applications and cannot be used to synchronize for coordinated actuation. In such a case, either always-on synchronization or a limited, *pre-facto* synchronization specifically designed for coordinated actuation is required. The concept of on-demand unites post- and pre-facto.

[14] Often called *skew* in the community.

[BD99, FOL01, FLO02].

An application that is currently starting to receive attention by industry and consumers is wireless home-media networking [Mos04, Dev03, Bar03, Bri03, Phi03]. Media content is made accessible to the living-room stereo system via an 802.11 link. This application is much simpler from the clock-synchronization perspective than the one we sketched in the previous section: Both stereo channels are transported as a unit to the stereo amplifier, while the distribution to the loudspeakers is still analog, and thus has a highly deterministic latency. The aforementioned digital distribution of individual audio channels is likely to become the next step in this development, as it is proposed e.g. by [Dic04].

Audio applications have also been studied by the wireless-sensor-networks community, either in the form of acoustic ranging [EGE02, MKSL04b] or more generally for a distributed audio-capturing platform [LKW03].

1.3.8 Discussion

In this section, we have argued that even though the basic problem of clock synchronization is intuitively simple, a wide range of different problems arise when looking at the details. It seems that clock synchronization has been a side issue in many different research areas and communities, while a common view is often missing. E.g. the contributions in the area of ad-hoc networks only seldom make use of previous results from work on fault-tolerant synchronization. A critical confrontation of practically achieved accuracy with the theoretical optimality results has, to our best knowledge, never been undertaken.

1.4 Thesis Outline

In the following, we summarize the contents and the main contributions of this thesis.

Chapter 2: Models and Methods
This chapter presents models and methods, which are used in Chapter 3, and which are extended in Chapter 4. These are

- Various alternative system models describing message delays and clock drift and an execution model for clock-synchronization algorithms (CSAs).
- Properties and metrics used in the analysis of CSAs.
- Techniques for measuring message delays and clock drift and measurement results from an 802.11b wireless LAN system.

- A framework for the optimization and comparison of CSAs, which is based on a combination of measurements and simulation. Evolutionary optimization is used for finding the parameters of a CSA.

Chapter 3: Point-to-Point Synchronization
This chapter studies the problem of synchronizing a drifting clock to a reference clock under the assumption of variable message delay. The novel class of Local-Selection CSAs is introduced and formally analyzed:

- Local-Selection CSAs exhibit properties which other CSAs do not, for example safety and liveness.

- Two upper bounds on the synchronization quality are derived. The first is based on the assumption of bounded delay, and meets a lower bound known from [LL84]. The second is based on the novel delay-interval model (presented in Chapter 2) and delivers a better bound. The delay-interval model can also provide estimates of how fast synchronization can be achieved.

- The requirements and benefits of drift compensation are analyzed under the assumptions of constant and of variable clock drift.

Heuristic variations of the Local-Selection CSAs are presented which make more economic use of memory and computation resources, and which require less knowledge about system parameters. Their performance is compared with that of other CSAs, using the wireless-loudspeakers application as a case study. It is demonstrated that the heuristic Local-Selection CSAs achieve a sufficient synchronization quality and are significantly more robust against cross traffic in the network.

Chapter 4: Multihop Synchronization
This chapter is joint work with Lennart Meier and studies clock synchronization in large-scale ad-hoc networks, for example wireless sensor networks. We present a modeling and analysis framework, which is applicable in a wide range of scenarios.

- A lower bound on the worst-case accuracy is derived and applied to tree-based CSAs.

- Interval-based synchronization is applied to the multihop synchronization problem in ad-hoc networks for the first time. It is shown that a simple algorithm from [MO83] is worst-case optimal.

- The interval-based and the tree-based approaches are compared in terms of worst-case accuracy and distribution of the energy consumption among the nodes in the network.

- The novel Back-Path algorithm is presented as an improvement of the algorithm from [MO83]. It is analyzed under which conditions the Back-Path algorithm outperforms the algorithm from [MO83], and the improvement is quantified using simulation of large-scale ad-hoc networks.

Chapter 5: Synthesis
In this chapter, we combine the results and insights from the previous two chapters. It is shown how the Local-Selection CSAs from Chapter 3 can be used to realize a communication event according to the zero-delay assumption from Chapter 4. The combination of the Local-Selection CSAs and the interval-based CSAs solves all three problems sketched in Sect. 1.1 and contributes to the implementation of a time-synchronization service in large-scale ad-hoc networks.

2

Models and Methods

The purpose of this chapter is to introduce the terminology, models and methods used in later chapters for the analysis of clock-synchronization algorithms (CSAs). Fig. 3 illustrates the structure of this chapter.

In Sect. 2.2, CSAs are formally defined. It is defined, what the input and what the output of a CSA is. Two simple but contrasting examples are presented. In Sect. 2.3, qualitative properties that formalize "reasonable" behavior of a CSA are presented. These properties can be evaluated in a system model that makes only minimal assumptions.

In Sect. 2.4, various metrics are introduced to assess the output of a CSA. In Sect. 2.5, formal system models are presented. These models state assumptions about message delays and clock drift, which allow us to make predictions about the worst-case bounds on the metrics defined in Sect. 2.4.

A more empirical approach to the evaluation of CSAs is presented in Section 2.6. The average-case performance of a CSA is evaluated based on system traces. It is shown how such traces can be obtained from a real system. We present data from an 802.11b wireless network, which will be used extensively in Chap. 3. In this section, we also discuss how traces can be generated from a formal system model. The generated traces are compared with the measured traces and the accurateness of the various system models is discussed.

In Section 2.7, we introduce multiobjective evolutionary algorithms as a tool for optimizing CSAs. Several trade-offs in the parameterization of CSAs are identified and the efficiency of the evolutionary approach is compared with standard search methods.

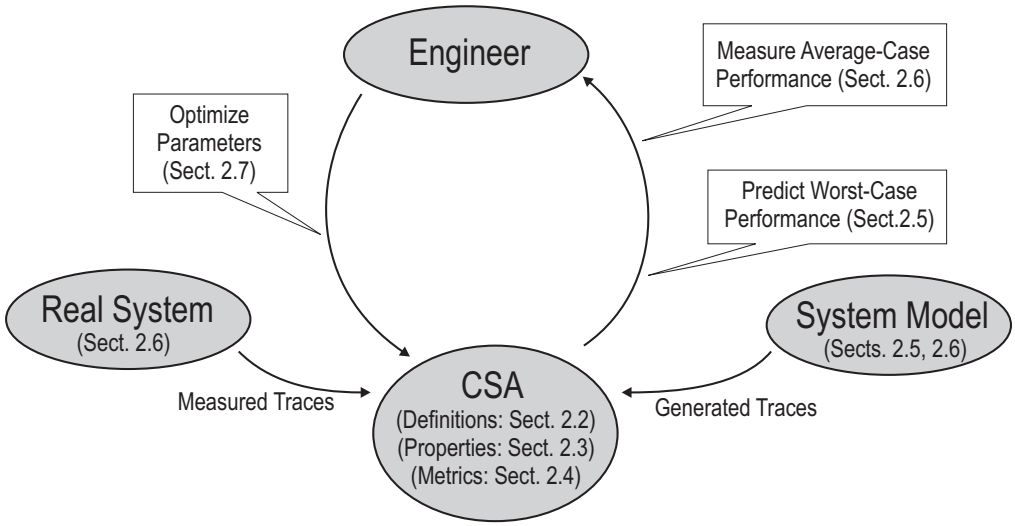


Fig. 3: *Overview of Chapter 2.* The thesis is concerned with the analysis of clock synchronization algorithms (CSAs), introduced in Sect. 2.2: (i) The worst-case performance is evaluated (Sect. 2.5) based on various assumptions about message delays and clock drift . (ii) The average-case performance is evaluated based on traces that have been measured in a real system or generated according to a particular system model (Sect. 2.6). As a fair evaluation requires that the parameters of a CSA are optimized, Sect. 2.7 presents an automated method to do so.

2.1 Notation

Capital latin letters are mainly used for functions, lower-case latin letters mostly represent scalars. Calligraphic letters are used for sets and algorithms. Greek letters usually are parameters of the system model or a CSA, most often they are scalars. We refer to some x at the time of the i-th message arrival (see next section) as x_i.

2.2 Algorithm Model

In this section, we formally define the notion of clock-synchronization algorithms (CSAs). The discussion is focused on *external* synchronization. This means that a clear distinction can be made between nodes that have access to reference time and nodes that do not have such information[1]. More precisely, we consider *point-to-point synchronization with unidirectional communication* from one reference node to one client node, which is illustrated in Fig. 4. The discussion of time synchronization is extended to arbitrarily complex topologies with multiple reference and client nodes in Chapter 4.

[1]Without loss of generality, we assume that reference time is real time.

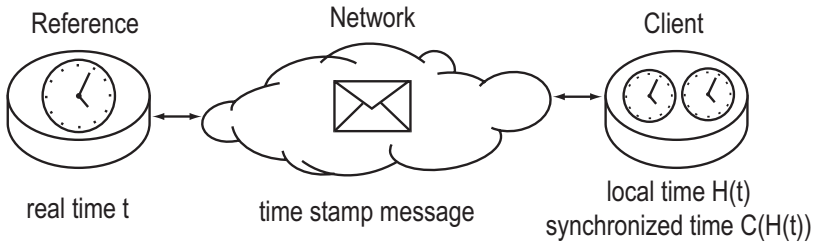


Fig. 4: *Model for point-to-point synchronization.* The system consists of a reference node, a client node, and a network. The reference node has access to real time, the client node has access to local time. The nodes can exchange messages containing time stamps. Time stamps are values of real time assigned to events occurring at the reference node and values of local time for events occurring at the client node. Events that are relevant for clock synchronization include message-send and message-receive events.

This section is organized as follows: First we introduce the notion of clocks, then we discuss how a CSA is executed and what kind of input it processes. Finally, we formally define what a CSA does and present two examples.

2.2.1 Clock Model

A clock is a physical or a virtual device that associates an event with a time. The time of a single event is called a time stamp and is a real number. Clocks are modeled as *piecewise continuous, strictly monotonic increasing functions*. We distinguish two types of clocks:

Def. 1: **(Local Clock)** *A local clock is a physical clock device, modeled as a function* $H : \mathbb{R} \to \mathbb{R}$. *We use the notation* $h_i = H(t_i)$ *to denote the local time* h_i *of an event* i *that occurs at real time* t_i. *The* drift *of a local clock is* $\rho(t) = \frac{\mathrm{d}H(t)}{\mathrm{d}t} - 1$. *The* drift variation *of a local clock is* $\vartheta(t) = \frac{\mathrm{d}\rho(t)}{\mathrm{d}t}$.

In general, the notion of *the local clock* is used to refer to the local clock of the client node. Note that the inverse function H^{-1} is the perfect solution to the time synchronization problem: To every time stamp h from the local clock, it associates the corresponding real time t. But this does not actually work, since the client does not know the function H.

Def. 2: **(Logical Clock)** *A logical clock is a function* $C : \mathbb{R} \to \mathbb{R}$. *It is a virtual clock, realized in software on a device that has a local clock. We use the notation* $c_i = C(h_i)$ *to denote the logical time* c_i *of an event* i *that occurs at* local time h_i. *The* drift *of a logical clock is* $\varrho(t) = \frac{\mathrm{d}C(H(t))}{\mathrm{d}t} - 1$.

Note that the argument of a logical-clock function always is a local time. A device that evaluates a logical clock knows only local time, but not real time. To refer to the logical time c_i of an event i that occurs at *real time* t_i, the notation $c_i = C(H(t_i))$ is used. The drift of a logical clock can also be written as $\varrho(t) = \frac{\mathrm{d}C(h)}{\mathrm{d}h}\frac{\mathrm{d}H(t)}{\mathrm{d}t} - 1 = \frac{\mathrm{d}C(h)}{\mathrm{d}h}(1 + \rho(t)) - 1$.

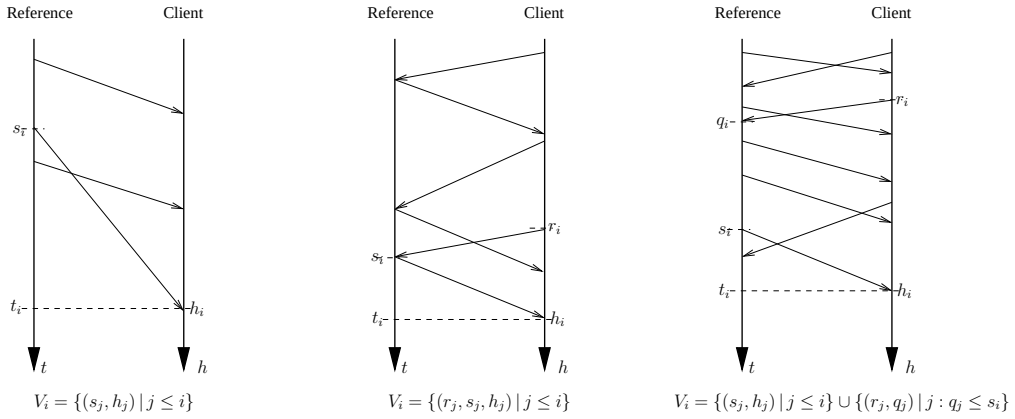


Fig. 5: *Three examples of views.* a) The reference node periodically sends a message containing the send time-stamp to the client node. b) The client node periodically sends a message containing the send time-stamp to the reference node, which immediately returns a message containing the received time stamp and the send time-stamp to the client node. c) Client and reference nodes append send time-stamps to messages that are exchanged between the nodes on behalf of other concurrent applications. The reference node additionally appends the latest received time stamp form the client node.

2.2.2 Input of CSAs

Clock-synchronization algorithms are executed locally on the client node. The goal is to compute a logical clock C that approximates the inverse local-clock function, i.e. $C \approx H^{-1}$. The computation of logical clocks is based on locally available information, which we call a *view*. Information that is relevant for synchronization is a set of time stamps. This set includes time stamps from the local clock, corresponding to locally observed events, and time stamps received in *time-stamp messages* from the reference node.

Def. 3: **(View)** *The view $\mathcal{V}_i$ is the set of all time stamps the client node has acquired up to and including local time h_i. The local time h_i is the local time when the i-th time-stamp message is received.*

The notion of views is illustrated in Fig. 5. The reference and the client node cooperate in exchanging time stamps. On the left, the view resulting from unidirectional communication from the reference node to the client node is shown. The reference node that repeatedly sends time-stamp messages to the client node, containing the send time stamp s_i. The time stamp h_i is the receive time according to the local clock of the client. The view $\mathcal{V}_i$ contains all pairs of send and receive time stamps (s_j, h_j) with $j \leq i$ [2].

In the middle and right drawings of Fig. 5, other views are shown, which include time-stamp messages from the client node to the reference node. The

[2]The index counts the time-stamp messages in the order of arrival at the client node. In the example depicted in Fig. 5, the $(i-1)$-th message has been sent after the i-th message, thus $s_{i-1} > s_i$. By convention, such an inversion can never occur for the receive time-stamps.

example in the middle represents the well-known *round-trip* synchronization scheme, e.g. used by the Cristian's algorithm [Cri89]. The following definitions can also be applied to this scheme, which we will do occasionally when comparing the unidirectional approach with the round-trip approach. However, the focus of this and the next chapter is on unidirectional synchronization, shown on the left. In Chapter 4, we will introduce CSAs that use the scheme shown on the right, which is a superposition of two uncoordinated unidirectional schemes between two peers.

2.2.3 Estimate-Based CSAs

We now introduce estimate-based CSAs. The predicate *estimate-based* is used to distinguish CSAs that compute estimates of real time from interval-based CSAs, which compute lower and upper bounds on real time and which are introduced in Chapter 4.

Def. 4: **(Estimate-Based CSA)** *An estimate-based clock-synchronization algorithm $\mathcal{A}$ computes from every view $\mathcal{V}_i$ a logical clock C_i. The* synchronized clock *is then as the function C, which at all local times $h \geq h_1$ is equal to the latest logical clock C_i:*

$$\begin{aligned} C_i &= \mathcal{A}(\mathcal{V}_i) \\ C(h) &= C_i(h)\ , \quad \text{with } i : h_i \leq h < h_{i+1} \end{aligned}$$

The synchronization error *at time t provided by algorithm $\mathcal{A}$ is*

$$E(t) = C(H(t)) - t \ .$$

We present two simple estimate-based CSAs which will be used throughout this chapter to illustrate concepts and definitions.

Def. 5: **(Local-Clock CSA)** *The algorithm $\mathcal{A}^{\text{loc}}$ is an estimate-based CSA that computes logical clocks according to*

$$C_i(h) = s_1 + (h - h_1)$$

Algorithm $\mathcal{A}^{\text{loc}}$ stores the first time-stamp pair (s_1, h_1) in memory and ignores all later time stamps. All logical clocks C_i are equal and the synchronized clock has a constant offset to the local clock.

Def. 6: **(Network-Clock CSA)** *The algorithm $\mathcal{A}^{\text{net}}$ is an estimate-based CSA that computes logical clocks according to*

$$C_i(h) = s_i + (h - h_i)$$

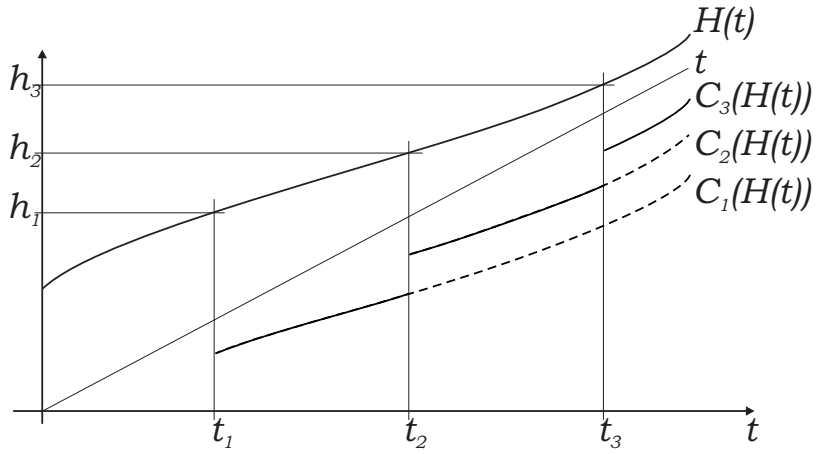


Fig. 6: *Logical clocks computed by* $\mathcal{A}^{\text{net}}$. The logical clocks C_1 – C_3 are shown as dashed lines, the local clock H and the synchronized clock C, which is always equal to the latest logical clock, are shown as solid lines.

Algorithm $\mathcal{A}^{\text{net}}$ only stores the latest time-stamp pair (s_i, h_i). The synchronized clock C leaps to s_i whenever a message is received. The offset to the local clock is constant only in between two time-stamp message arrivals. Figure 6 illustrates the behavior of algorithm $\mathcal{A}^{\text{net}}$.

The algorithms $\mathcal{A}^{\text{loc}}$ and $\mathcal{A}^{\text{net}}$ represent two extreme cases. The first mostly relies on the local clock, while the latter is mostly based on the received time stamps. As we will see later, both algorithms have advantages and disadvantages and good CSAs are often a compromise between these extremes.

In the next chapter, a particular kind of estimate-based CSAs will play an important role:

Def. 7: **(Selective CSA)** *An estimate-based CSA is a selective CSA if, for every view* $\mathcal{V}_i$*, it computes (i) a candidate clock* C_i^* *, (ii) a decision* $\pi_i \in \{0, 1\}$*, and (iii) a logical clock according to*

$$C_i(h) = \begin{cases} C_i^*(h) & \textit{if } \pi_i = 1 \\ C_{i-1}(h) & \textit{if } \pi_i = 0 \end{cases}$$

If $\pi_i = 1$, then the candidate clock is selected and $C_i(h) = C_i^*(h),\ \forall h \geq h_i$. If $\pi_i = 0$, the candidate clock is rejected and $C_i(h) = C_{i-1}(h),\ \forall h \geq h_i$, i.e. the new logical clock is equal to its predecessor.

Clearly, every estimate-based CSA is selective in some sense. However, we reserve the term for CSAs that compute non-trivial decisions π_i. Algorithm $\mathcal{A}^{\text{loc}}$ has $\pi_1 = 1$ and $\pi_i = 0$ for all $i > 1$ and is thus not a selective CSA. Also algorithm $\mathcal{A}^{\text{net}}$ with $\pi_i = 1$ for all i is not a selective CSA.

2.3 Properties

In the previous section we have defined what CSAs are. But what is a good CSA? This question is the topic of the next four sections, and we will argue that the answer strongly depends (i) on the exact requirements and (ii) on the system in which the CSA is executed. But first, we present a way of analyzing CSAs that allows us to make statements about CSAs *without making any assumptions about the systems in which the CSAs are executed*. This analysis is based on properties that formalize correct behavior on a microscopic level, i.e. on properties that describe the processing of a *single view* $\mathcal{V}_i$. To this purpose, we define the error $e_i = C_i(h_i) - t_i$ as the synchronization error *immediately after* starting the logical clock C_i, and $e_i^- = C_{i-1}(h_i) - t_i$ as the synchronization error *immediately before* starting the logical clock C_i.

Def. 8: **(Safety)** *An estimate-based CSA $\mathcal{A}$ is said to be safe if the error immediately after starting a new logical clock is never worse than it was immediately before starting this clock:*

$$\forall i > 1 : |e_i| \leq |e_i^-|$$

Ex. 4: *Algorithm $\mathcal{A}^{\text{loc}}$ is safe: It never modifies the synchronized clock after the first time-stamp message arrival and thus $e_i = e_i^-$ for all $i > 1$.*

Algorithm $\mathcal{A}^{\text{net}}$ is not safe: Consider the case when the client's local clock has drift $\rho = 0$, the first time-stamp message delay is 1ms, and the second is 2ms. Then $e_2^- = -1\text{ms}$ and $e_2 = -2\text{ms}$. Thus $|e_2| > |e_2^-|$ and $\mathcal{A}^{\text{net}}$ is not safe.

We have said before that the properties do not require to make assumptions about the system. But to show that $\mathcal{A}^{\text{net}}$ is not safe, we have made such assumptions. This is no contradiction: If a property is required to hold in all possible systems, we can construct an arbitrary system to show that the property does not hold.

While $\mathcal{A}^{\text{loc}}$ is safe, we would not say that it is a very good CSA, since it achieves safety by simply never doing anything (for $i > 1$). Therefore we define the liveness property:

Def. 9: **(Liveness)** *An estimate-based CSA $\mathcal{A}$ is said to be live, if on an infinite sequence of views, no logical clock C_i is the last that has a better synchronization error than its predecessor.*

$$\forall i : \exists j > i : |e_j| < |e_j^-|$$

Ex. 5: *The algorithm $\mathcal{A}^{\text{loc}}$ is not live, since $e_i = e_i^-$ for all $i > 1$.*

The algorithm $\mathcal{A}^{\text{net}}$ is not live: Consider the case when all time-stamp message delays are 1ms. Then the error is $e_i = -1\text{ms}$ for all i. If the client's local clock drift is positive ($\rho > 0$), but time-stamp messages are received sufficiently often such that $e_i^- < 0$, then $e_i < e_i^- < 0$ and thus $|e_i| > |e_i^-|$ for all i.

None of the two CSAs introduced so far is safe *and* live. In the next chapter, we will present CSAs that fulfill both properties. These algorithms never actively degrade the synchronization error and *eventually* improve the synchronization error, thus we can say that they are *correct*. In the following, we present an alternative definition of correctness that applies to selective CSAs only:

Def. 10: *A selective CSA is optimally selective if it never makes a wrong decision. Let $e_i^* = C_i^*(h_i) - t_i$ be the initial error of the candidate clock C_i^*. Then a selective CSA is optimally selective if*

$$\begin{aligned} |e_i^*| > |e_i^-| \quad &\rightarrow \quad \pi_i = 0 \\ |e_i^*| < |e_i^-| \quad &\rightarrow \quad \pi_i = 1 \end{aligned}$$

Optimal selectivity implies safety (by condition $|e_i^*| > |e_i^-| \rightarrow \pi_i = 0$). Instead of requiring that the CSA eventually improves the synchronization error, it must improve *whenever possible*.

2.4 Performance Metrics

In this section, we are interested in quantitative statements about CSAs. The question is not whether a CSA is good or not, which was the focus of the previous section, but rather *how good* it is. Furthermore, we do not focus our attention on a single execution of a CSA (processing a single view $\mathcal{V}_i$), but consider a synchronized clock C over an extended period of interest.

Let the observation interval $T = [t^s, t^e]$, starting at real time t^s and ending at real time t^e, and its length $\Delta T = t^e - t^s$. It does not make sense to observe the synchronization error of a clock before the synchronization algorithm has been started, thus $t^s \geq t_1$ (remember that t_1 is the time of the first time-stamp message arrival). However, the observation may be started at $t^s > t_1$ if one is interested in the steady-state behavior only. In this observation interval, we define the following three performance metrics:

Def. 11: **(Accuracy)** *The accuracy A of a synchronized clock C is the maximal absolute value of the synchronization error:*

$$A = \max_{t \in T} \left(|E(t)| \right)$$

Def. 12: **(Peak Jitter)** *The peak jitter J of a synchronized clock C is the difference between the maximal and the minimal value of the synchronization error:*

$$J = \max_{t \in T} \left(E(t) \right) - \min_{t \in T} \left(E(t) \right)$$

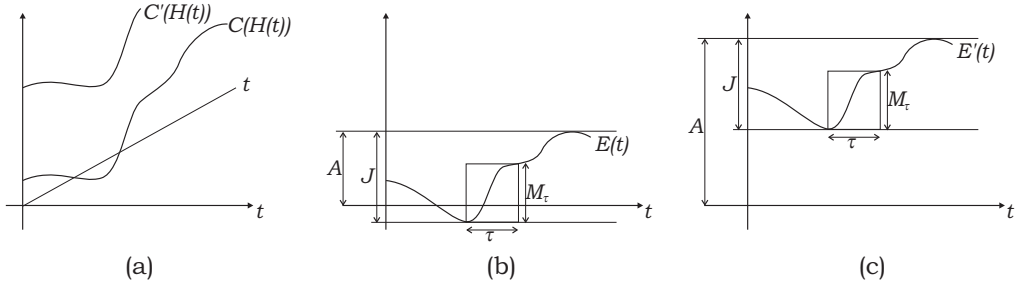


Fig. 7: *Performance metrics accuracy A, peak jitter J, and MTIE M_τ.* (a) Two synchronized clocks C and C'. (b) Error of C and corresponding performance metrics. (c) Error of C'. Here the accuracy is larger than the peak jitter.

Def. 13: **(Maximum Time-Interval Error (MTIE))** *The maximum time-interval error M_τ of a synchronized clock C is the maximal difference in the synchronization error within an observation interval of real-time length τ:*

$$M_\tau \;=\; \max_{t^s \le t \le t^e - \tau} \left(\max_{t \le u \le t+\tau} (E(u)) - \min_{t \le u \le t+\tau} (E(u)) \right)$$

The accuracy A describes how large the absolute synchronization error can become, the peak jitter J describes the variability of the synchronization error. The MTIE M[3] is similar to the peak-jitter metric. It also measures the variability of the synchronization error, but not over the whole observation period, only over a smaller interval. Note that $M_{\tau_1} \le M_{\tau_2}$ if $\tau_1 \le \tau_2$ and $J = M_{\Delta T}$. Figure 7 illustrates the three metrics, a concrete example how the various metrics can be used to model the synchronization requirements of an application can be found in Sect. 3.6.

For the three performance metrics defined above, we have implicitly used a target setup time of $\tilde{S} = t^s - t_1$, i.e. we have not considered the interval $[t_1, t^s]$[4]. We will now define the actual setup time S as a performance metrics, using target values for the other three metrics.

Def. 14: **(Setup Time)** *The setup time S of a synchronized clock C is defined as the smallest real number such that the metrics A, J and M in the observation interval $T = [t_1 + S, t^e]$ meet their target values, i.e. $A \le \tilde{A}$, $J \le \tilde{J}$ and $M \le \tilde{M}$.*

We now return to the initial question of this section: How good are algorithms $\mathcal{A}^{\text{loc}}$ and $\mathcal{A}^{\text{net}}$? The question can for example be answered by determining the worst possible accuracy, peak jitter or MTIE these algorithms achieve.

[3] We will omit the interval length τ whenever it is clear from the context or irrelevant.

[4] We will sometimes use the notation $J(\tilde{S})$ to refer to the peak jitter after a setup time of $\tilde{S}$.

2.5 Worst-Case Performance

To determine the worst-case performance of a CSA, it has to be specified what the worst case is. The system can be interpreted as an *adversary* that knows exactly how a CSA works and then generates a sequence of views $\mathcal{V}_i$ that makes the CSA behave badly. In this section, we specify the system model, i.e. the rules that the adversary has to obey when generating worst-case views.

In our model of point-to-point synchronization with unidirectional communication, the adversary can determine (i) the drift of the client's local clock and (ii) the delays of the time-stamp messages. In Sect. 2.5.1, we present clock-drift models, and in Sects. 2.5.2 and 2.5.3, two different delay models are introduced.

2.5.1 Clock Drift

In the whole thesis, we always assume local clocks with bounded drift.

Ass. 1: **(Bounded Drift)** *The drift $\rho(t)$ of a local clock lies at all times t in the interval $[-\hat{\rho}, \hat{\rho}]$.*

The bounded-drift model is the most general and the most pessimistic model. It allows the speed of local clocks to make arbitrary jumps in the interval $[-\hat{\rho}, \hat{\rho}]$. Real clock devices do not behave like this. It is however a suitable model to determine the worst-case performance. Sometimes, we will make stronger assumptions about the drift of local clocks.

Ass. 2: **(Bounded Drift Variation)** *The variation $\vartheta(t) = \frac{\mathrm{d}\rho(t)}{\mathrm{d}t}$ of a local clock's drift lies in the interval $[-\hat{\vartheta}, \hat{\vartheta}]$.*

Note that bounded drift variation is an additional constraint for the adversary and never replaces the bounded-drift assumption. A special case of bounded drift variation is constant drift, i.e. when $\hat{\vartheta} = 0$, then $\rho(t) = \rho$ is constant. The constant drift model with $\vartheta = 0$ is too optimistic in practice. It will be used in Sect. 3.4.1 to develop a drift-compensation mechanism, which in Sect. 3.4.2 is extended to the bounded-drift-variation model.

2.5.2 Bounded Delay

We start with a very simple, standard delay model. It is assumed that time-stamp messages are received regularly and that their delay is bounded by finite constants.

Ass. 3: **(Message Interval)** *In every interval of real-time length Δt, at least one time-stamp message is received by the client node.*

$$t_i - t_{i-1} \leq \Delta t, \quad \forall i > 1$$

Ass. 4: **(Bounded Delay)** *The delay d_i of every time-stamp message is bounded by the finite constants $d_{\min}$ and $d_{\max}$.*

$$d_i \in [d_{\min}, d_{\max}], \quad \forall i \geq 1$$

	Worst case		Ex. WLAN		Ex. WAN	
	$\mathcal{A}^{\mathrm{loc}}$	$\mathcal{A}^{\mathrm{net}}$	$\mathcal{A}^{\mathrm{loc}}$	$\mathcal{A}^{\mathrm{net}}$	$\mathcal{A}^{\mathrm{loc}}$	$\mathcal{A}^{\mathrm{net}}$
A	$d_{\max} + \hat{\rho}\Delta T$	$d_{\max} + \hat{\rho}\Delta t$	$110\mathrm{ms}$	$10,01\mathrm{ms}$	$1100\mathrm{ms}$	$1001\mathrm{ms}$
J	$\hat{\rho}\Delta T$	$d_{\mathrm{var}} + 2\hat{\rho}\Delta t$	$100\mathrm{ms}$	$9,02\mathrm{ms}$	$100\mathrm{ms}$	$990,2\mathrm{ms}$
M_τ	$\hat{\rho}\tau$	$d_{\mathrm{var}} + \hat{\rho}\tau$	$0,1\mathrm{ms}$	$9,01\mathrm{ms}$	$0,1\mathrm{ms}$	$990,1\mathrm{ms}$

Tab. 2: *Worst-case performance of $\mathcal{A}^{\mathrm{loc}}$ and $\mathcal{A}^{\mathrm{net}}$ in the bounded-delay model:* The result of a performance comparison depends on the applied metric, on system parameters like the maximal drift $\hat{\rho}$, bounds on the message delay $d_{\min}$ and $d_{\max}$, and the message interval Δt, and on the length of the observation interval ΔT.

Occasionally, we use the notation $d_{\mathrm{var}} = d_{\max} - d_{\min}$.

Now, it is finally possible to determine how good algorithms $\mathcal{A}^{\mathrm{loc}}$ and $\mathcal{A}^{\mathrm{net}}$ perform in the worst case.

Ex. 6: *The synchronization error achieved by algorithm $\mathcal{A}^{\mathrm{loc}}$ is $E(t) = s_1 + H(t) - h_1 - t = -d_1 + (H(t) - h_1) - (t - t_1) = -d_1 + \int_{t_1}^{t} \rho(t)\mathrm{d}t$. The worst-case peak jitter is thus $J = \hat{\rho}\Delta T$ if $\rho = \pm\hat{\rho}$. The worst-case accuracy is $d_{\max} + \hat{\rho}\Delta T$ if $d_1 = d_{\max}$ and $\rho = -\hat{\rho}$. The worst-case MTIE is $M_\tau = \hat{\rho}\tau$.*

The synchronization error achieved by algorithm $\mathcal{A}^{\mathrm{net}}$ is $E(t) = -d_i + \int_{t_i}^{t} \rho(t)\mathrm{d}t$ with $t_i \leq t < t_{i+1}$. $\mathcal{A}^{\mathrm{net}}$ can achieve at most $-d_{\min} + \hat{\rho}\Delta t$ and at least $-d_{\max} - \hat{\rho}\Delta t$. The worst-case peak jitter is thus $J = d_{\mathrm{var}} + 2\hat{\rho}\Delta t$. The worst-case accuracy is $A = d_{\max} + \hat{\rho}\Delta t$ and the worst-case MTIE is $M_\tau = d_{\mathrm{var}} + \hat{\rho}\tau$.

These results are summarized in Tab. 2. Two numerical examples are also shown in this table. We use $\tau = 1\mathrm{s}$, $\Delta T = 1000\mathrm{s}$, $\hat{\rho} = 10^{-4}$. For the wireless LAN example, $\Delta t = 100\mathrm{ms}$, $d_{\min} = 1\mathrm{ms}$, and $d_{\max} = 10\mathrm{ms}$. For the WAN example $\Delta t = 10\mathrm{s}$, $d_{\min} = 10\mathrm{ms}$, and $d_{\max} = 1\mathrm{s}$.

We see that $\mathcal{A}^{\mathrm{loc}}$ in both examples achieves a better MTIE than $\mathcal{A}^{\mathrm{net}}$. On the other hand, $\mathcal{A}^{\mathrm{net}}$ achieves the better accuracy. Concerning the peak jitter, $\mathcal{A}^{\mathrm{loc}}$ is better in the WAN scenario, and $\mathcal{A}^{\mathrm{net}}$ is better in the WLAN where the message delay is less variable and time-stamp messages are received more frequently.

The example has shown that the decision whether $\mathcal{A}^{\mathrm{loc}}$ or $\mathcal{A}^{\mathrm{net}}$ is better depends on the requirements: $\mathcal{A}^{\mathrm{net}}$ often provides a better accuracy, while $\mathcal{A}^{\mathrm{loc}}$ promises a lower MTIE. Which one provides the lower peak jitter depends essentially on the length of the observation interval ΔT. Whether $\mathcal{A}^{\mathrm{loc}}$ or $\mathcal{A}^{\mathrm{net}}$ is better also depends on system characteristics: Algorithm $\mathcal{A}^{\mathrm{net}}$ is very sensible to the variability d_{var} of the time-stamp message delays, while $\mathcal{A}^{\mathrm{loc}}$ depends strongly on the maximal rate deviation $\hat{\rho}$ of the local clock. Finally, the decision depends on the available resources: Algorithm $\mathcal{A}^{\mathrm{net}}$ also requires that time-stamp messages arrive frequently (small Δt). Some of the worst-case bounds presented in Table 2 are presumably very pessimistic, e.g. the bound for the MTIE in the case of algorithm $\mathcal{A}^{\mathrm{net}}$, which implicitly assumes that the time-stamp messages with the shortest and the longest delay are received consecutively. Also, some of the bounds are either difficult to determine in advance or have to be chosen very pessimistically. E.g. the maximal delay in a WLAN can

achieve several hundreds of milliseconds, while most of the delays are below two milliseconds. To summarize, the value of the presented worst-case bounds is rather dubious in practice.

2.5.3 Delay-Interval Curves

The bounded-delay model presented above constrains every message delay to lie in a given interval, independent of previous delays. We now introduce a slightly more complex delay model that constrains every message delay to lie in an interval whose size depends on previously encountered message delays. This alternative delay model is original work and will allow us to derive smaller upper-bounds on performance metrics than the simple bounded-delay model in Sect. 3.3.

The basic idea is the following: The bounded-delay model requires that in every interval of length Δt, a message with a delay in $[d_{\min}, d_{\max}]$ is received. We extend this with a second constraint, saying that in every interval of length $2\Delta t$, at least one message with a delay in $[d_{\min}, D^u(2)]$ is received. Similarly, in every interval of length $j\Delta t$, at least one message is received that has a delay in $[d_{\min}, D^u(j)]$. We call the function $D^u : \mathbb{N} \rightarrow \mathbb{R}$ the upper delay-interval curve.

Ass. 5: **(Delay-Interval Model)** *A sequence $(d_i),\ i \geq 1$ of message delays is an admissible sequence in the delay-interval model if in every interval of length Δt, at least one time-stamp message is received, and if in every interval of length $j\Delta t$ with $j \geq 1$, at least one message has a delay not greater than $D^u(j)$.*

$$\max_{i \geq 1} \left\{ \min_{k \in [i, i+j-1]} \{d_k\} \right\} \leq D^u(j), \forall j$$

Figure 8 shows two examples of delay-interval curves. In the following, we discuss (i) how an upper delay-interval curve can be obtained if a delay sequence is given, and (ii) how delay sequences can be generated if an upper delay-interval curve is given.

Computation of the Delay-Interval Curves

Given a sequence (d_i), we want to compute an upper delay-interval curve such that the sequence (d_i) is admissible. There exist infinitely many such curves, e.g. $D^u(j) = \infty$ for all $j \geq 1$ is always a solution, but does not really capture the properties of the sequence (d_i). Instead,

$$D^u(j) = \max_{i \geq 1} \left\{ \min_{k \in [i, i+j-1]} \{d_k\} \right\} \tag{2.1}$$

provides the smallest function D^u for which the sequence (d_i) is admissible. We call this function the tight delay-interval curve. It is a monotonic decreasing function. Note that the value of $D^u(1)$ is equal to the maximal delay in the sequence. Inversely, $D^u(\infty)$ is the minimal delay.

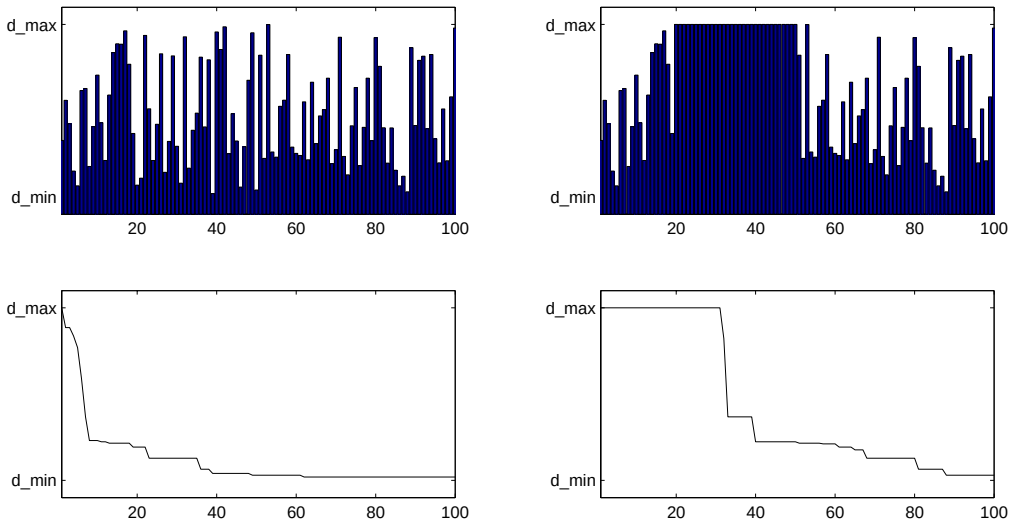


Fig. 8: *Two examples of delay-interval curves.* The curves are computed from the delay sequences using Eq. 2.1. Both sequences are equal except that in the sequence on the right, 30 consecutive delays are maximal.

Algorithm 1 Generate Delay Sequence

Input: Upper delay-interval curve D^u of length π
Output: Admissible delay sequence (d_i) with length I

for $i := 1$ **to** I **do**
  $\mathrm{DMAX} := D^u(1)$
  for $j := 1$ **to** $\min(\pi, i-1)$ **do**
    if $\min_{k\in[i-j,i-1]}(d_i) > D^u(j)$ **then**
      $\mathrm{DMAX} := \min(\mathrm{DMAX}, D^u(j))$
    end if
  end for
  $d_i :=$ random number in $[D^u(\pi), \mathrm{DMAX}]$
end for

Generation of the Delay Sequences
Deriving an admissible delay sequence (d_i) from a given delay-interval curve can be done using Alg. 1. The last statement of the algorithm can be changed to $d_i := \mathrm{DMAX}$ to obtain the most pessimistic admissible delay sequence. Also, it could be enhanced to account for a particular delay distribution function.

2.6 Average-Case Performance

In the last section, we have presented analytical methods to derive worst-case bounds on the performance achieved by a CSA. The results of this analysis are *guaranteed* under the assumptions made in the system model. From a prac-

titioner's point of view, there are two problems with this approach: (i) Real systems, though nasty as they are, do not behave as adversaries. They do not take pleasure in leading the engineer and his algorithms astray, and thus the analytical worst-case bounds are often far too pessimistic. (ii) Analytical system models make many abstractions and simplifications on the actual behavior of a real system. System models describe what a scientist would like a system to be, such that he can analyze it nicely. Therefore, the worst-case "guarantees" presented in the last section are by no means guaranteed in a real system.

In this section, we present an alternative way to evaluate the performance of CSAs, based on evaluating *system traces* and *measuring* the performance according to the metrics defined in Sect. 2.4. In Sect. 2.6.1, we present a method to obtain system traces by measurement in a real system and discuss empirical date recorded in an 802.11b WLAN. Alternatively, traces can be generated according to an analytical system model. In Sect. 2.6.2, we present statistical models that allow to generate traces and determine their parameters according to the measured traces. In Sect. 2.6.3, we confront the performance obtained by simulation of the well-known Linear-Regression CSA on measured and on generated traces.

A view contains all the information that a client node can use for synchronization. However, it does not contain sufficient information for assessing the synchronization quality. For this purpose, we define the notion of a *trace*.

Def. 15: **(Trace)** *A Trace $\mathcal{T}_i$ contains the information of the view $\mathcal{V}_i$ augmented with the real times t_i of the time-stamp-message arrivals at the client node:*

$$\mathcal{T}_i = \mathcal{V}_i \bigcup \{t_j \,|\, j \leq i\} \quad .$$

Using traces, we can derive the delays of time-stamp messages from the reference to the client node. Since s_i is the real time of the send event and t_i is the real time of the receive event, we have $d_i = t_i - s_i$. It is also possible to derive the clock drift of the client node's local clock. More precisely, the average clock drift between two time-stamp-message arrivals is $\rho_i = (h_i - h_{i-1})/(t_i - t_{i-1}) - 1$.

2.6.1 Measured Traces

Here we describe how traces can be obtained from a real system and present measured probability-distribution functions for various scenarios and load conditions.

We record traces consisting of 50000 time-stamp messages. For every message, the send time stamps s_i, the local receive time stamps h_i and the *virtual* receive time-stamp t_i have to be measured. We call t_i a virtual time stamp, because it is a value of the sender's clock at an event that occurs at the receiver. Measuring t_i is thus the most difficult part of recording system traces.

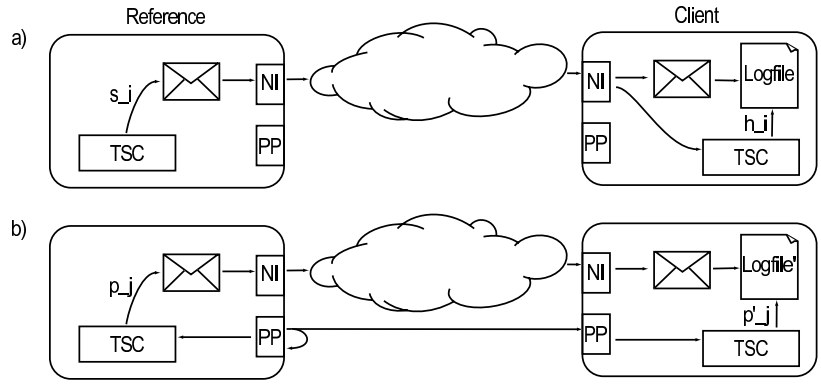


Fig. 9: *Setup for the delay measurements.* a) The send and receive time stamps s_i and h_i are measured with the local TSC. b) The virtual receive time t_i is interpolated using local time stamps p_j and p'_j from parallel-port interrupts, which are generated concurrently on both nodes.

Measurement System
Nodes are standard PCs running the Linux operating system. The nodes use their time-stamp counter (TSC) register, which exists in every Pentium processor, to measure the time of events. Its resolution is determined by the clock frequency of the CPU, i.e. the TSC of a Gigahertz Pentium has a resolution of one nanosecond. On the reference node, a periodic process generates time-stamp messages. The send time stamps s_i are written into these messages. When a message is received at the client node, the send time stamp is written into a log together with the corresponding receive time stamp h_i.

Measuring the Virtual Receive Time t_i
The virtual receive time t_i is equal to the local receive time h_i plus the current offset between the reference's and the client's TSC. This offset can be measured as follows: A direct cable connects the parallel ports of the reference and the client nodes. The reference node periodically generates falling and rising edges on one of its parallel-port output pins. This pin is connected to the external interrupt pin of the parallel ports of the reference and the client node. Thus, whenever the reference generates a rising edge on its parallel-port output pin, simultaneous interrupts are generated on both nodes. Reference and client record their TSC registers in the parallel-port interrupt service routine. The reference sends its parallel-port time stamp to the client, which writes it together with its own local parallel-port time stamp in a second log. Using interpolation methods, it is now possible to compute virtual receive times t_i of the timestamp-messages according to the TSC register of the reference. The procedure is illustrated in Fig. 9

Accuracy of the Measurements
All time-stamping mechanisms are implemented in loadable kernel modules in order to reduce OS latencies that otherwise would cause additional delay vari-

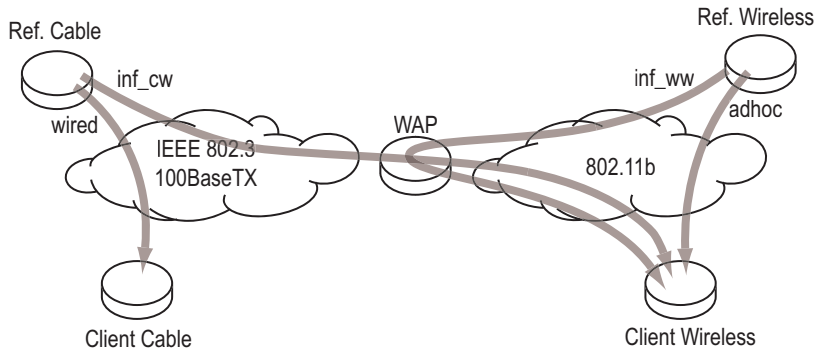


Fig. 10: *System setup for recording traces. wired*) The reference and the client nodes are in the wired part of the setup, *inf_cw*) reference is wired, the client wireless, *inf_wc*) reference is wireless, the client is a wired node, and *inf_ww,adhoc*) both reference and client are wireless nodes.

ability. A detailed description of the measurement system is given in [BD03], similar procedures are described in [MFNT00] and [EGE02]. The accuracy of the measurements has been found to be better than $\pm 5\mu s$.

Scenarios
Figure 10 displays the various scenarios for which we record traces. An IEEE 802.11b wireless network is connected via an access point (WAP) to a switched 100Base-TX IEEE 802.3 Ethernet. Various combinations of the reference and the client node in either the wireless or the wired part of the network are examined. In the case of both reference and client node being wireless nodes, the 802.11b network is configured both in infrastructure and in ad-hoc mode.

Network Load
For all the scenarios depicted in Fig. 10, we have recorded several traces under variable conditions. Several types, directions and rates of network load have been generated during the measurements. Figure 11 shows all scenarios with a wireless client (i) without any additional network load, (ii) with load generated by a 128kBit/s, constant-rate MP3 audio-stream, and (iii) with load generated by a 3MBit/s, variable-rate DivX video-stream. The MP3 and the DivX streams were generated by the reference node[5] and received by the client node. Detailed results are shown in Table 3, many more scenarios and load conditions have been presented in [BD03].

Discussion
As can be seen in Tab. 3, the minimal delay is different for the various scenarios. In the infrastructure mode with wireless reference and client nodes, the minimal delay is twice as large as the minimal delay in the ad-hoc mode, which could be expected since the messages are first sent from the reference node to

[5]Using the Video LAN Client (VLC) software for Linux.

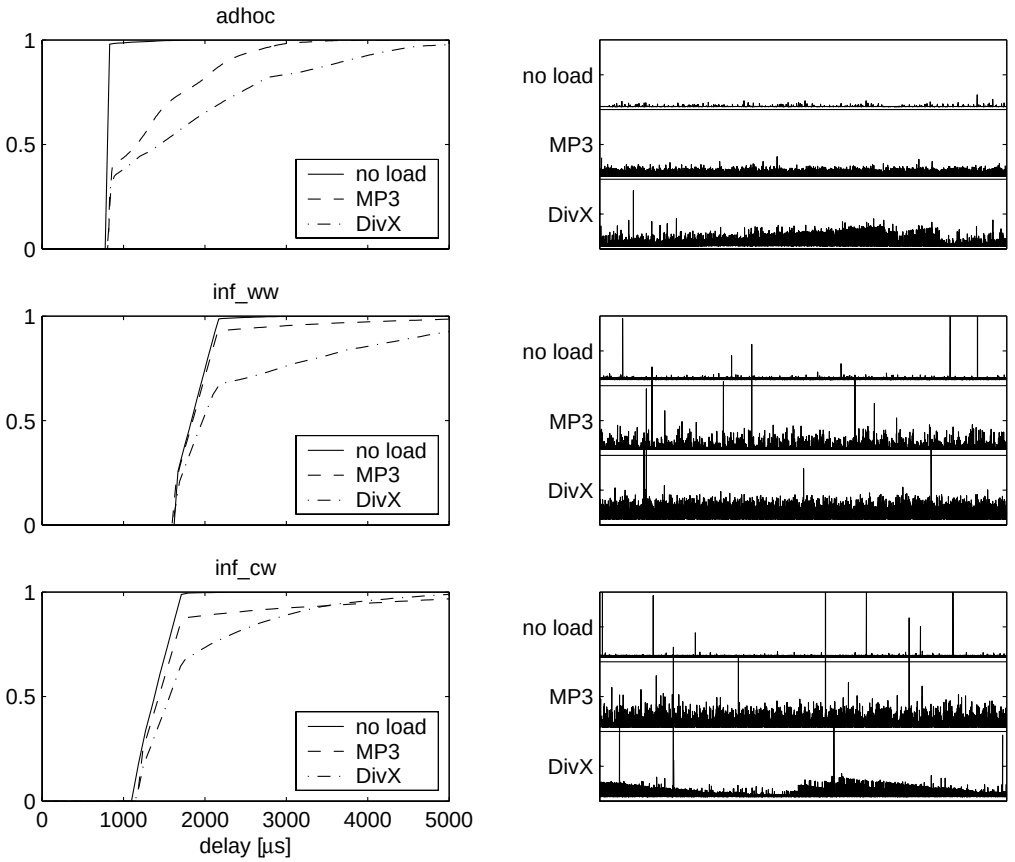


Fig. 11: *Measured cumulated probability-density functions (CDFs) and delay sequences.* On the left, measured probability functions for different scenarios with a wireless client and different load conditions are shown. On the right, corresponding delay sequences are shown. All sequences are shown in the same scale to allow a qualitative comparison.

the WAP and then from the WAP to the client node. If the client node is a wired node, the minimal delay is shorter, since the access point retransmits the message on the much faster wired network. In the opposite direction, the minimal delay is slightly higher, while for the purely wired scenario, the minimal delay is almost two orders of magnitude smaller. *In all scenarios, the minimal delay remains almost constant (variations below $24\mu\mathrm{s}$) under the various load conditions.* This observation is the main reason why the Local-Selection CSAs presented in Chapter 3 achieve a better performance, especially in heavy and variable network conditions, than CSAs that do some form of averaging over many time stamps (see Sect. 3.6).

Increasing network load has a much stronger influence on the median delay $d_{0.5}$ and the average delay d_{avg}. The effect is most prominent in the case of the ad-hoc mode, where the median delay increases by more than $600\mu\mathrm{s}$ under heavy network load and the average delay increases even by more than $1\mathrm{ms}$.

In all scenarios and under all load conditions, the maximal delay can achieve

Scenario	*Load*	$d_{\min}[\mu s]$	$d_{0.25}[\mu s]$	$d_{0.5}[\mu s]$	$d_{0.75}[\mu s]$	$d_{\max}[\mu s]$	$d_{\text{avg}}[\mu s]$
adhoc	none	811	830	827	830	53533	837
	light	800	841	1167	1737	7221	1365
	heavy	809	834	1458	2449	17559	1850
inf_ww	none	1638	1669	1858	2035	45737	1888
	light	1636	1698	1875	2053	50683	2017
	heavy	1637	1771	2014	2923	82349	2605
inf_wc	none	1143	1216	1220	1226	3317	1225
	light	1125	1007	1217	1007	671187	7247
	heavy	1121	1169	1212	2231	212460	10081
inf_cw	none	1207	1273	1420	1624	87800	1459
	light	1216	1316	1472	1692	75181	1770
	heavy	1192	1347	1592	2109	50812	1914
wired	none	24	32	34	35	151	33
	light	20	35	41	47	302	41
	heavy	22	98	135	167	425	134

Tab. 3: *Statistical properties of the measured traces.* For every scenario and load type, at least 10 traces have been recorded. The values shown above are the averages of these traces.

very large values ($> 50\text{ms}$). As can be seen in Fig. 11 on the right side, it is mainly the WAP that occasionally causes very large message delays.

2.6.2 Generated Traces

After describing how to measure traces, we present an alternative way of obtaining traces. Message delays can be modeled as a random variable whose distribution $F(d) = \text{Probability}(d_i \leq d)$ is specified. We review three different probability functions that have been used in the literature and discuss their applicability to the scenarios described above.

For example, a normal distribution according to

$$F^N(d) = \frac{1}{2}\left(1 + \text{erfc}(\frac{d - d_{\text{avg}}}{\sigma\sqrt{2}})\right)$$

can be assumed, where σ is the standard deviation of the delay. In some instances, the normal distribution is a good description of the reality, e.g. the delay of the low-level RF channel of Berkeley Motes has been reported to closely follow a normal distribution [EGE02]. Normal distributions are often assumed for simulation [vGR03, OS94] and the derivation of optimality results [KEES03]. However, the normal distribution is symmetric and does not model a minimal delay. This is often not realistic, e.g. in the case of WANs [Cri89, Tro94, Fie04].

An exponential-distribution model according to

$$F^E(d) = \begin{cases} 0 & \text{if } d < d_{\min} \\ 1 - e^{\frac{d_{\min} - d}{d_{\text{avg}} - d_{\min}}} & \text{otherwise} \end{cases}$$

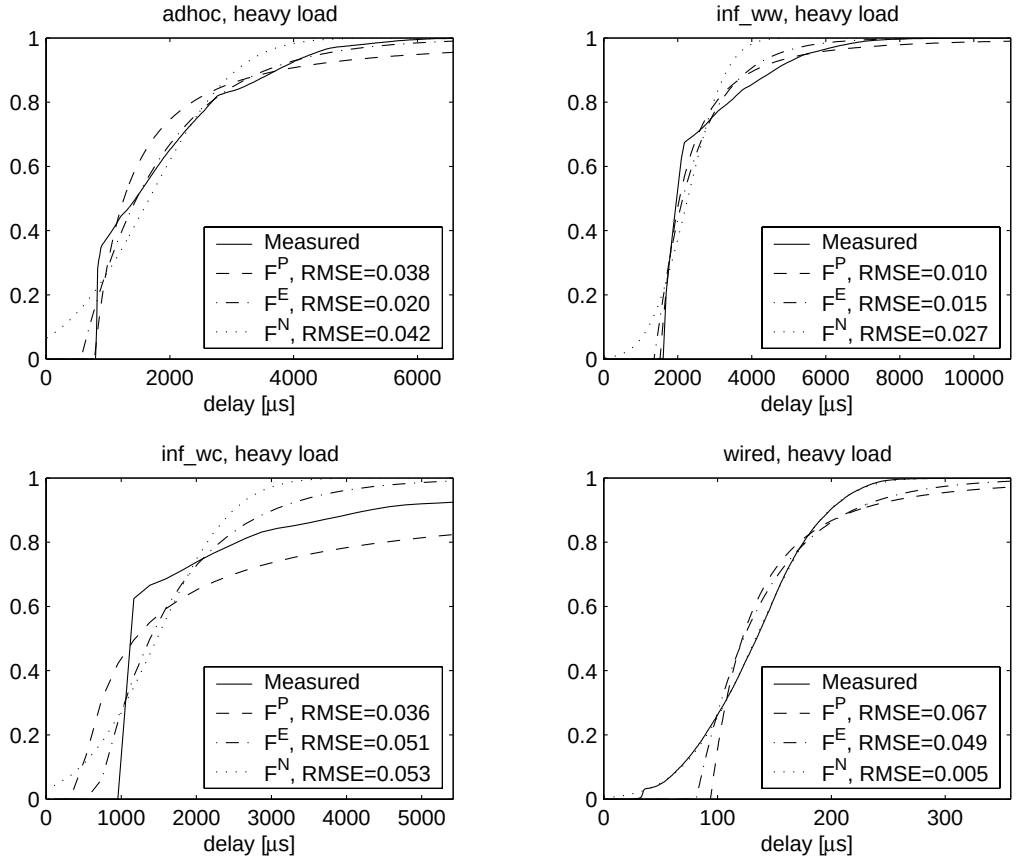


Fig. 12: *Measured CDF and best-fitting models.* P is a Pareto model, E exponential and N normal. RMSE is the root mean square error of the model; small values indicate good fit. The exponential model fits best in the ad-hoc scenario, the Pareto is best in the infrastructure scenarios, and the normal model fits best in the wired-Ethernet scenario. None of the models is very accurate, as can be seen most clearly in the infrastructure, wireless-to-cable scenario (inf_wc).

may be more accurate in these cases. It has been found that exponential distributions often are too optimistic in the sense that the probability of delays exceeding the expected delay d_{avg} is underestimated.

Heavy-tail distributions describe such situations more accurately [Fie04]. An example is the Pareto distribution defined as

$$F^P(d) \;=\; \begin{cases} 0 & \text{if } d < d_{\min} \\ 1 - \left(\frac{d_{\min}}{d}\right)^{\alpha} & \text{otherwise} \end{cases}$$

Discussion

We have fitted the three probability functions to the measured probability functions using the root mean square error (RMSE) as the optimization criterion.

Scenario	*Load*	*Pareto*		*Exponential*		*Normal*	
		$d_{min}[\mu s]$	$\alpha[1]$	$d_{min}[\mu s]$	$d_{avg}[\mu s]$	$d_{avg}[\mu s]$	$\sigma[\mu s]$
adhoc	none	776	57.8	856	861	801	10
	light	768	2.0	682	1388	1283	583
	heavy	783	1.5	562	1864	1668	1100
inf_ww	none	1634	7.6	1622	1859	1848	201
	light	1605	5.9	1610	1923	1867	231
	heavy	1522	2.3	1369	2479	2272	831
inf_wc	none	1210	127.0	1210	1220	1218	8
	light	335	1.9	450	693	730	208
	heavy	430	0.7	678	1695	1476	791
inf_cw	none	1163	5.5	1156	1422	1392	203
	light	1153	3.5	1155	1531	1460	246
	heavy	1173	2.5	1100	1825	1684	570
wired	none	31	13.1	31	34	33	2
	light	34	4.9	33	42	41	7
	heavy	94	2.7	81	141	133	51

Tab. 4: *Parameters of the best-fitting models for various scenarios and network-load conditions.* The parameters are significantly different for all scenarios and load conditions.

Table 4 shows the parameters of the best-fitting models, Fig. 12 displays the measured probability functions and the best-fitting models for scenarios with heavy load (DivX). In Tab. 4, it can be seen that the parameters are significantly different for the various scenarios and have to be individually determined for every scenario. Fig. 12 shows that no single statistical model fits best in all scenarios: For the wired case, the normal distribution is most accurate (smallest RMSE), the exponential model is best in the case of the ad-hoc scenario, and the Pareto model is best in the other scenarios. Remember that we have noted in the discussion of the measured probability functions that the ad-hoc scenario has less outliers than corresponding infrastructure scenarios. It is thus not surprising that the exponential model is strongest in the ad-hoc scenario.

A downside of statistical delay models is that they assume statistical independence of concurrent delays. As will be seen in Sect. 2.6.3 , this leads to far too optimistic performance estimations using artificially generated delay sequences, even if their probability function fits well the measured probability function.

2.6.3 Experimental Study: Accuracy of Delay Models

In this section we have presented several possibilities of obtaining traces: (i) Record from a real system, (ii) generate delays using measured CDFs, (iii) generate delays using a statistical delay model, (iv) generate delays using measured delay-interval curves. We now examine how well the various models capture the properties of the real system regarding clock synchronization.

Experimental Setup

We compare the MTIE in an interval of length $\tau = 10\text{s}$, achieved by the Linear-Regression CSA on traces generated with the various models. The Linear-Regression CSA is explained in Example 7. The window size was set to $\kappa := 500$. We have computed the average probability function from ten recorded traces containing 10000 time-stamp messages. From the resulting probability function we have generated ten traces with the same length. Then we have fitted normal, exponential, and Pareto probability functions, and again generated ten traces from all these models (using the constant drift model). Finally, we computed the average-delay curves from the ten recorded traces and generated ten traces from these curves. The complete procedure was executed with recorded traces from the ad-hoc scenario and the infrastructure, wired reference, and wireless client scenario.

Ex. 7: **(Linear-Regression CSA $\mathcal{A}^{\text{llr}}$)** *Algorithm $\mathcal{A}^{\text{net}}$ (Def. 6) starts every logical clock C_i with the received time stamp as the initial value. Therefore time-stamp messages with very large delays cause large deviations in the synchronization error. The linear-regression CSA smoothes such deviations by averaging several received time stamps to produce initial values of the logical clocks. The linear-regression algorithm is shown in Alg. 2. The algorithm maintains four sums S_{loc}, S_{ref}, S_{loc2} and S_{locref} which range over the last κ time-stamp messages. With these sums, the algorithm computes offset and slope of the regression line. The algorithm minimizes the sum of the square deviations between the last κ time stamps and the regression line.*

Algorithm 2 Linear-Regression CSA, following [PTVF92], Section 15.2

Input: View $\mathcal{V}_i = \{(h_j, s_j) | j \leq i\}$
Output: Logical clock C_i
Parameters: Averaging-window size κ
State: $S_{\text{loc}} := S_{\text{ref}} := S_{\text{loc2}} := S_{\text{locref}} := 0$

$S_{\text{loc}} := S_{\text{loc}} + h_i$
$S_{\text{ref}} := S_{\text{ref}} + s_i$
$S_{\text{loc2}} := S_{\text{loc2}} + h_i^2$
$S_{\text{locref}} := S_{\text{locref}} + h_i * s_i$
if $i > \kappa$ **then**
  $S_{\text{loc}} := S_{\text{loc}} - h_{i-\kappa}$
  $S_{\text{ref}} := S_{\text{ref}} - s_{i-\kappa}$
  $S_{\text{loc2}} := S_{\text{loc2}} - h_{i-\kappa}^2$
  $S_{\text{locref}} := S_{\text{locref}} - h_{i-\kappa} * s_{i-\kappa}$
end if
if $i > 1$ **then**
  $C_i(H(t)) := \frac{S_{\text{loc2}} S_{\text{ref}} - S_{\text{loc}} S_{\text{locref}}}{S_{\text{loc2}} - S_{\text{loc}}^2} + H(t) \frac{S_{\text{locref}} - S_{\text{loc}} S_{\text{ref}}}{S_{\text{loc2}} - S_{\text{loc}}^2}$
else
  $C_i(H(t)) := (s_i - h_i) + H(t)$
end if

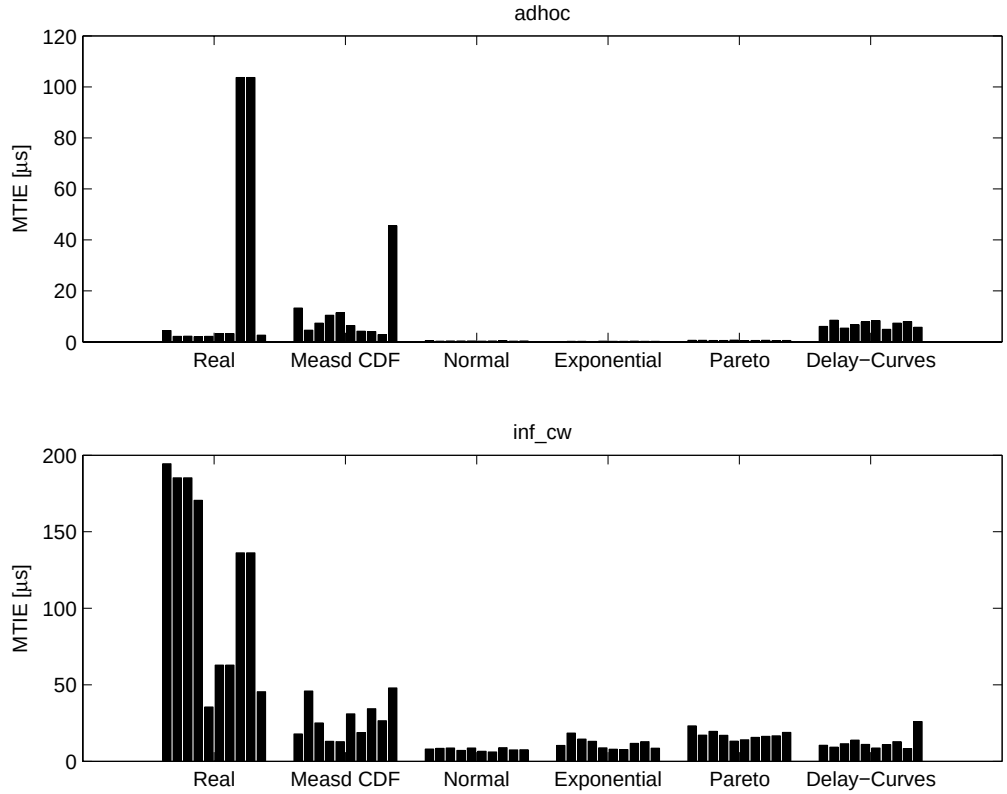


Fig. 13: *MTIE achieved by the Linear-Regression CSA on recorded traces compared to the values achieved on generated traces.* The upper chart shows the results for the ad-hoc scenario; the lower chart shows those of the infrastructure scenario with a wired reference and a wireless client. On the left, the MTIE achieved on ten recorded traces is shown, followed by the MTIE achieved on traces generated on the base of a given probability function. The rightmost bars show the results achieved on traces generated with the delay-curves model. The MTIE values achieved on normal and exponential traces for the ad-hoc scenario are too small (i.e. too good) to be visible.

Linear-regression mechanisms are used e.g. in the RBS synchronization algorithm for sensor networks [EGE02]. In [Arv94], the peak jitter achieved by a simpler version of the linear-regression algorithm is analyzed using statistical methods and assuming that statistical properties of the message-delay sequence are known (expected delay, standard deviation). In contrast to the linear-regression algorithm presented here, the algorithm of [Arv94] starts a new logical clock only every κ arrivals of a time-stamp message. Under these assumptions, a method for choosing the parameter κ is presented. Note that $\mathcal{A}^{\mathrm{llr}}$ requires $O(\kappa)$ memory.

Discussion

Figure 13 shows all the resulting MTIE values. It can be seen that all models correctly identify the infrastructure scenario to be more difficult than the ad-hoc

scenario. In the ad-hoc scenario, $\mathcal{A}^{\mathrm{llr}}$ achieves much better results on statistical models than on the recorded traces. The traces generated with the measured probability function and with the delay-curves produce results that are close to those achieved on the recorded traces. In the infrastructure scenario, there is only a small difference between the statistical models and the delay-curve model. The normal-distribution model leads to the most optimistic MTIE values, while the exponential and the Pareto models, which generate more and larger outliers, give worse and more variable results. In summary, all artificially generated traces lead to too optimistic results and fail to deliver realistic estimations of the synchronization performance in a specific, real system. Therefore, we base our quantitative evaluations on measured traces.

2.7 Parameter Optimization

In this section, we argue that the parameter optimization of realistic CSAs is a difficult task, and show that multiobjective evolutionary algorithms (MOEAs) are a well-suited method. In two experimental studies, conflicting goals in the design of CSAs are identified, and the performance of MOEAs in comparison with more traditional optimization methods is evaluated.

2.7.1 Motivation

Not all CSAs are as simple as the Local-Clock and the Network-Clock CSAs (see Defs. 5 and 6). For example the performance of the linear-regression CSA presented in Example 7 presumably depends on the averaging-window size κ, set to $\kappa = 500$ in the experiments of Sect. 2.6.3. A larger κ would probably have reduced the damaging influence of time-stamp messages with a very large delay, so-called outliers. On the other hand, averaging over a long period, aside from costing a lot of memory, is problematic because of non-constant clock drift, since then the assumption of a linear relation between local time and reference time is not justified. The best choice of κ is thus a compromise that is dependent on the frequency and the amplitude of outliers and on the speed and amount of change in the clock drift. Maybe it would be a better idea to enhance $\mathcal{A}^{\mathrm{llr}}$ by an outlier-rejection mechanism: if a received time stamp is more than a specified amount off the current synchronized-clock time, the time-stamp message is discarded instead of being included in the average calculations. Such a mechanism introduces at least one more parameter, for which it is again not clear how it is chosen optimally and what its impact on the choice of the averaging-window size is.

We see that CSAs that aim at achieving robust performance in a real system tend to have relatively many parameters and determining their optimal settings is not trivial. As will be seen in the next chapter, mechanisms for automatic parameter adaption most often have parameters themselves, and are thus no real

solution to the parameter-optimization problem. Furthermore, optimal parameter settings for one scenario are not necessarily optimal in another scenario. The problem is especially grave if the performance of several different CSAs is compared: Assume two CSAs $\mathcal{A}^1$ and $\mathcal{A}^2$; irrespective of the algorithms' mechanisms, it is very likely that one can show that $\mathcal{A}^1$ is better than $\mathcal{A}^2$, if one tunes $\mathcal{A}^1$'s parameters carefully while no effort is spent on the parameters of $\mathcal{A}^2$. For all these reasons, a method for *automatic* parameter optimization is required, which allows to clearly define the effort that is spent on optimizing a CSA's parameters.

Many standard optimization methods are not applicable, as the relation between a given parameter and the algorithm's performance can not be described analytically, e.g. gradient-search methods which require differentiability of the performance metrics. Also, many applications require CSAs that simultaneously optimize several performance metrics. E.g. the wireless-loudspeakers application sketched in Fig. 2 relies on an accuracy in the order of a millisecond, a peak jitter in the order of 100 microseconds, an MTIE with $\tau = 10s$ in the order of 10 microseconds and a setup time in the order of a few seconds. Many standard optimization methods can only optimize a single criterion, which forces the user to weigh the various criteria a priori. This is often difficult, if nothing is known about achievable values. Methods that concurrently optimize several criteria are much more user-friendly in such a situation, as they clearly identify which combinations are achievable and thus allow an educated trade-off decision after the optimization step.

We are thus looking for a method that allows us to automatically optimize a large number of parameters according to multiple, maybe conflicting criteria, and which does not require differentiability of the relation between parameters and the optimization criteria.

2.7.2 Multiobjective Evolutionary Optimization

We now present the principles of multiobjective evolutionary optimization algorithms (MOEAs), which fulfill the requirements described above.

Evolutionary algorithms are based on the notions of *individuals* and *populations*. An individual is a representation of a potential solution to the optimization problem, and a population is a set of individuals. Every individual is characterized by its genotypic and a phenotypic representation. For example the genotype of a CSA are its parameters, while the phenotype of a CSA is its performance achieved on a given trace or set of traces. From the genotypic representation, the phenotypic representation can be obtained by simulation of the CSA with the given parameters. This process is called *fitness assignment*.

Evolutionary algorithms work on populations. Starting with an *initial population*, an iterative process generates new populations, called *generations*. A new generation is derived from its predecessor as follows: In a first step, the *environmental selection* removes a given number of individuals from the population, namely those that have a comparably low fitness. Other criteria may

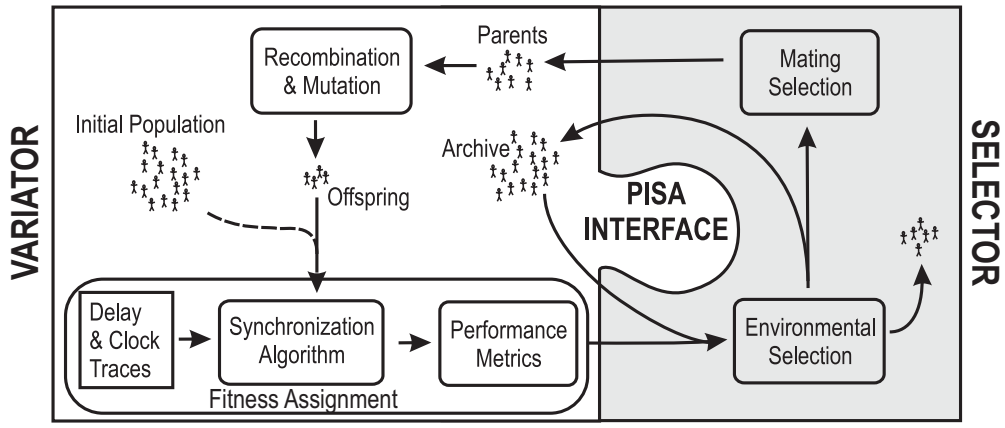


Fig. 14: *Evolutionary optimization process as supported by the PISA tool set.* A parameter set for a CSA constitutes an individual. The variator component (left-hand side) computes the fitness of a population of individuals. The selector component (right-hand side) decides which individuals are not part of the next generation of individuals and which individuals are parents for the next generation. The variator then recombines and mutates the parameters of the parent individuals to create the offspring. The process is repeated from the beginning.

also influence this step, e.g. the goal of preserving diversity in the population. In a second step, the *mating selection* chooses some individuals from the population that will serve as parents. In the final step, new individuals are added to the population to compensate for the environmental selection. This is done by *recombination* of two parent-individuals, i.e. a new individual is created that shares some of its genotypic representation with each of its parents, and applying *mutation*, i.e. some random modification of a part of the genotypic representation. The set of all new individuals is called *offspring*.

The PISA Interface

As illustrated by Fig. 14, the components of an MOEA can be divided into two components. (i) The problem-dependent functionality, which includes recombination and mutation of the individuals and fitness assignment, is grouped into the *variator* component. (ii) The problem-independent functionality is grouped in the *selector* component. This component decomposition has the advantage that a user only has to implement the variator component and can use tested standard implementations of the selector component. The freely available PISA interface [BLTZ03] provides easy access to a variety of state-of-the-art MOEAs. In our experimental studies, we used the SPEA2 algorithm [ZLT02].

Variator for CSA Optimization

We now describe in more detail the implementation of the variator component.

Individuals: The genotypic representation of an individual consists of the sequence of a CSA's parameters. Depending on the type of CSA, the parame-

ters can be integers or reals or a mix of both. E.g. for the linear-regression CSA presented in Example 7, it is a single integer value, the window size κ. The phenotypic representation is a set of performance metrics achieved by the CSA on a given trace or set of traces. To avoid an over-specialization of the parameters, it is advantageous to simulate the CSA on several traces and compute the performance metrics as the worst values achieved on any of the traces.

Recombination: Two parent individuals A and B are recombined to create two new individuals C and D by randomly selecting an integer number i and setting the first i parameters of the individual C equal to those of parent A and the remaining parameters equal to those of parent B. The new individual D has the first i parameters from B and the others from A.

Mutation: For every new individual, a randomly chosen parameter is multiplied with a real number chosen randomly from $[0.5, 1.5]$. For efficiency, we have implemented the compute-intensive simulation of the CSAs in C code, while the recombination and mutation functionality is written in Matlab[6].

2.7.3 Experimental Study: Conflicting Criteria

In the previous description, the selection of the performance metrics as optimization objectives has not been specified. The following experimental study has the purpose of illustrating the use of MOEAs for parameter optimization and of identifying two pairs of conflicting objectives. We introduce the phase-locked-loop CSA $\mathcal{A}^{\mathrm{pll}}$, which is particularly well suited for this purpose, since it has only a small number of parameters and the reason for the conflicting objectives are intuitively accessible.[7]

Ex. 8: **(Phase-Locked-Loop CSA)** *Algorithm $\mathcal{A}^{\mathrm{loc}}$ (Def. 5) starts the first logical clock C_1 with an initial value equal to the first time stamp s_1 and ignores all later time stamps. The phase-locked-loop algorithm $\mathcal{A}^{\mathrm{pll}}$ shown in Alg. 3 is similar in that only the first time stamp s_1 is used as an initial value for the logical clock started at its arrival. The following time stamps are used to adjust the rates of later logical clocks. Whenever a received time stamp s_i has a larger value than the current reading of the latest logical clock $C_{i-1}(h_i)$, the new logical clock C_i is set to progress faster than the previous logical clock. The rate adjustment is determined by a PI controller with the difference $s_i - C_{i-1}(h_i)$ as input. The offset of the new logical clock C_i is determined such that the synchronized clock C is continuous, i.e. such that $C_i(h_i) = C_{i-1}(h_i)$. A possible implementation is shown in Alg. 3. An additional feature of this implementation is the limitation of the input for the PI controller to $\pm\theta_{\max}$, which limits the negative impact of outliers. Various forms of PLL algorithms are used in ATM (synchronous AAL, see [Nor00], Sec. 2.3). Also NTP contains a PLL [Mil95].*

[6] On a 2x900MHz SunBlade 1000, the fitness assignment based on 3 traces of length 50000 for a population of 40 individuals takes less than 8 seconds, a complete run with 100 generations takes 13 minutes.

[7] Another reason is that, together with the linear-regression CSA, $\mathcal{A}^{\mathrm{pll}}$ will be used as a yardstick when we evaluate the performance of a new family of CSAs in the next chapter.

Algorithm 3 Phase-Locked-Loop CSA

Input:	View $\mathcal{V}_i = \{(h_j, s_j) \vert j \leq i\}\mathcal{V}_i$
Output:	Logical clock C_i
Parameters:	Integral gain κ_I
	Proportional gain κ_P
	Maximal input signal θ_max
State:	Integrator sum $S_\mathrm{I} := 0$
Temporary variables:	Input signal θ

if $i > 1$ **then**
  $\theta := s_i - C_{i-1}(h_i)$
  if $\theta > \theta_\mathrm{max}$ **then**
    $\theta := \theta_\mathrm{max}$
  end if
  if $\theta < -\theta_\mathrm{max}$ **then**
    $\theta := -\theta_\mathrm{max}$
  end if
  $S_\mathrm{I} := S_\mathrm{I} + \kappa_\mathrm{I}(h_i - h_{i-1})\theta$
  $C_i(H(t)) := C_{i-1}(h_i) - \frac{h_i}{1+\kappa_\mathrm{P}\theta+S_\mathrm{I}} + H(t)\frac{1}{1+\kappa_\mathrm{P}\theta+S_\mathrm{I}}$
else
  $C_i(H(t)) := (s_i - h_i) + H(t)$
end if

Experimental Setup

A population of 20 individuals representing $\mathcal{A}^{\mathrm{pll}}$ algorithms is optimized during 20 generations for two objectives. The objectives are (i) the conflicting objectives MTIE M_τ and peak jitter J with a setup time of $30\mathrm{s}$, and (ii) the conflicting objectives peak jitter and the setup time S, measured at the time when peak jitter is better than $300\mu\mathrm{s}$ and the MTIE is below $10\mu\mathrm{s}$. The whole procedure is performed for the ad-hoc and the wireless infrastructure scenario, both with all three load types.

Discussion

From every optimization, the two extremal solutions, i.e. those with the best MTIE and the best peak jitter, respectively the best peak jitter and the shortest setup time are displayed in Tab. 5. As an example, consider the case of the infrastructure scenario with heavy load (DivX). The first optimization obtained an algorithm that achieves after $30\mathrm{s}$ a peak jitter of $427\mu\mathrm{s}$ and an MTIE of $16\mu\mathrm{s}$. The other algorithm achieves the much worse peak jitter of $1929\mu\mathrm{s}$ but a better MTIE of $2\mu\mathrm{s}$. The second optimization obtained one algorithm that achieves a peak jitter of $400\mu\mathrm{s}$, but requires $1495\mathrm{s}$ to reduce the peak jitter to the target value of $300\mu\mathrm{s}$. The other algorithm achieves a peak jitter of $1030\mu\mathrm{s}$ after $30\mathrm{s}$, but can reduce it to $300\mu\mathrm{s}$ after $103\mathrm{s}$. The trade-offs are less pronounced for the experiments with no or light load.

Figure 15 illustrates what the trade-off between peak jitter and MTIE actually means regarding the algorithm's behavior. While optimizing the peak jitter

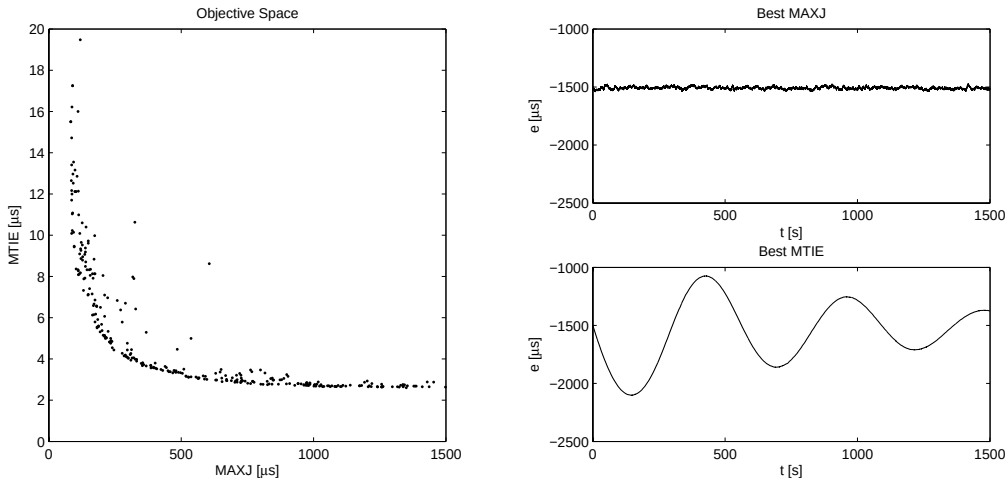


Fig. 15: *Optimization criteria can be conflicting.* On the left, the Peak jitter and MTIE values achieved by the PLL algorithm optimized using SPEA2. On the right, the synchronization error over time of the two extremal solutions is shown.

results in a PLL with a very short time constant (high proportional gain, low integral gain), optimizing the MTIE results in a longer time constant. An algorithm with a long time constant is more resilient to outliers, but is very slow in settling to a steady state. Returning to the wireless-loudspeakers application from Fig. 2, the interpretation is the following: Optimizing peak-jitter results in a noisy performance, while optimizing MTIE results in a solution where the perceived direction of the sound's source is not constant.

The study showed that the various performance metrics defined in Sec. 2.4 can indeed be conflicting objectives regarding parameter optimization. Multiojbective optimization allows us to first explore the range of feasible trade-offs, and to weigh the various objectives afterwards.

Scenario	*Load*	*Best* $J[\mu s]$		*Best* $M_\tau[\mu s]$		*Best* $J[\mu s]$		*Best* $S[s]$	
		J	M_τ	J	M_τ	J	S	J	S
adhoc	none	7	2	26	1	6.40	0	6.40	0
	light	163	17	1336	6	156.40	1000	337.04	32
	heavy	1813	57	10235	8	1850.96	1000	14745.53	783
inf_cw	none	63	11	2714	3	61.04	6	94.26	0
	light	74	11	1442	2	72.00	1119	90.73	0
	heavy	427	16	1929	2	400.31	1495	1030.16	103

Tab. 5: *Conflicting objectives.* Performance metrics achieved by the PLL algorithm for a given scenario and load condition. Every double column (Best X) represents one parameterization. The first two parameterizations are obtained by optimizing the objective values peak jitter J (Def. 12) and MTIE M_τ in an interval of length $\tau = 10\text{s}$ (Def. 13), the last two algorithms are obtained by optimizing the peak jitter and the setup time S (Def. 14).

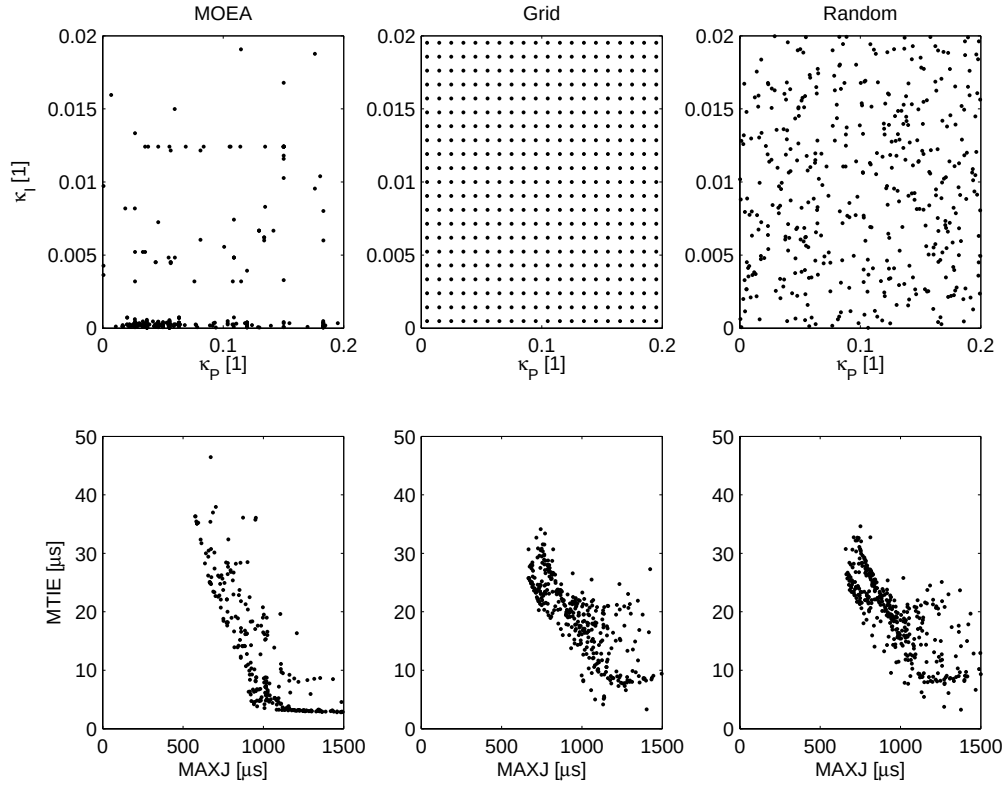


Fig. 16: *Comparison of parameter-optimization algorithms.* Parameter optimization using SPEA2, grid search and random search. For all methods, 2500 solutions were evaluated. The upper charts display two of the parameters of the evaluated candidate solutions, the lower charts display the corresponding objective values.

2.7.4 Experimental Study: MOEAs vs. Random and Grid Search

Having shown that multiobjective optimization is a reasonable approach to parameter optimization of CSAs, we now compare the proposed evolutionary algorithm with two simple alternatives: grid search and random search.

Experimental Setup

An initial population of 50 individuals of phase-locked-loop algorithms was optimized using the MOEA during 50 generations. The objective values were for peak jitter and MTIE and the traces were taken from the infrastructure scenario with a wired reference (inf_cw). In total, the MOEA evaluated 2500 candidate parameter sets. The solutions found by the MOEA are compared to solutions found by the grid-search and the random-search methods, both under the constraint of evaluating the same number of candidate parameterizations.

Discussion

Figure 16 displays the parameterizations of $\mathcal{A}^{\text{pll}}$ that were evaluated by the three optimization methods. The grid- and the random-search methods explore more

or less uniformly the whole parameter space and achieve a very similar coverage in the objective space. The MOEA concentrates its search on a limited area of the parameter space. While the solution with the best MTIE is comparable to the respective solutions found by grid and random search, the MOEA finds solutions with a better peak jitter (500μs vs. 592μs and 601μs). Also the region of interesting trade-offs is much more densely populated in the objective space of the MOEA. The area dominated by the solutions found by the MOEA (with the corner point $J = 2000\mu$s and $M_\tau = 100\mu$s) is 1.3 times larger than the area dominated by the solutions found by grid search and 1.45 times larger than the corresponding area for random search. These factors are not particularly impressing. However in the next chapter, we will see CSAs that have up to nine parameters. To systematically explore a nine-dimensional parameter space using grid search is almost infeasible: Evaluating 2500 solutions would mean that for every parameter, only 2 or 3 values could be examined. In contrast, the MOEA quickly finds interesting areas in the parameter space and concentrates its efforts on them. Concerning the running time, all optimization algorithms are approximately equal, since the running time is strongly dominated by the fitness assignment (simulation of the CSA on the input traces), which is the same for all of the optimization algorithms.

2.8 Summary

In this chapter, methods to evaluate clock-synchronization algorithms (CSAs) have been presented. Some of the methods are more theoretical, based on formal system models. Others are empirical, based on measurements in a real system. In the next chapter, both approaches will be applied to the novel class of Local-Selection CSAs.

We have presented properties of CSAs that formalize correct or reasonable behavior and which can be evaluated without a complete specification of the system model. While these properties allow us to reason about the qualities and weaknesses of a particular CSA, they do not give any quantitative information about the CSAs performance. Therefore, various metrics have been introduced to quantify the performance of a CSA. We have seen that it is necessary to make detailed assumptions about clock drift and message delays to derive worst-case bounds on these metrics.

We have presented a method to evaluate the average-case performance of CSAs using simulation. Various ways to obtain system traces for the simulation have been compared. It has been shown that artificially generated traces lead to performance estimations that are too optimistic. Multiobjective evolutionary algorithms (MOEAs) have been presented as a tool for optimizing the parameters of a CSA. Combining measurements, simulation and optimization provides for a realistic, reproducible and fair evaluation and comparison of different CSAs, as illustrated in Fig. 17.

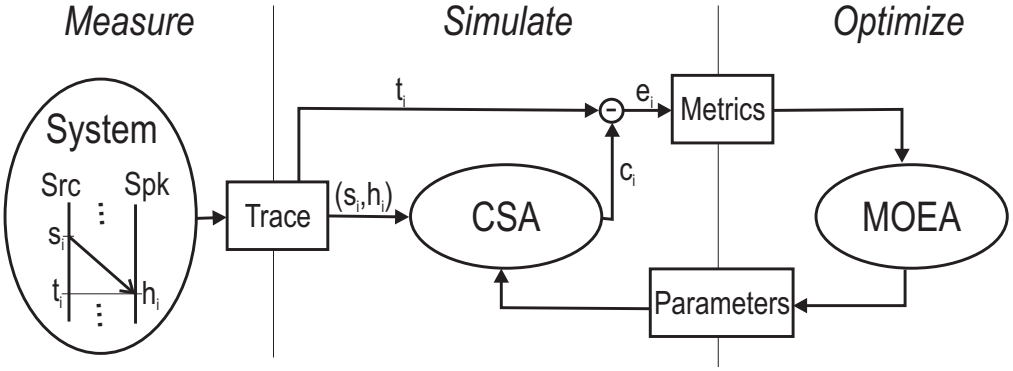


Fig. 17: *Realistic, reproducible and fair evaluation strategy. Measure:* Send and receive time stamps s_i and h_i are stored together with reference time t_i. *Simulate:* CSAs are simulated on the trace, producing estimates c_i. With the reference time, the synchronization error e_i can be determined. *Optimize:* The parameters of the CSA are optimized using a multiobjective-optimization evolutionary algorithm (MOEA).

In contrast to direct measurements of the synchronization quality in a real system, the presented method is reproducible, because it uses simulation based on recorded traces. In contrast to simulation based on artificially generated traces, it is realistic, since the traces have been obtained by measurements in a real system, accounting for side effects like OS latencies, cross traffic in the network, etc. The method is fair, because parameter optimization of the CSAs is an integral part of the evaluation. The method will be applied to the wireless-loudspeakers application in the next chapter.

3

Point-to-Point Synchronization

In this chapter, the synchronization of a single client node to a reference node is discussed. It is assumed that clock-synchronization algorithms (CSAs) run locally on the client and process a sequence of time-stamp messages received from the reference node, without sending messages to the reference node. The novel class of Local-Selection CSAs (LS-CSAs) is introduced. In Sects. 3.2–3.4, LS-CSAs are formally analyzed, contributing to a better understanding of the achievable performance of point-to-point synchronization. In Sects. 3.5 and 3.6, the theoretical insights of previous sections are translated to implementable, efficient algorithms. The algorithms are evaluated using simulation based on measured traces. It is demonstrated that LS-CSAs outperform known CSAs and reliably achieve synchronization in the low microsecond range in 802.11b WLANs.

In Sect. 3.1, the defining properties of the LS-CSAs are introduced. The decisive property of *slow logical clocks* and its ramifications are illustrated by an informal comparison with a similar CSA from Lamport presented in [Lam78].

In Sect. 3.2, it is shown that LS-CSAs are safe, live, and optimally selective. These properties guarantee a reasonable behavior of a CSA, independent of assumptions on message delays. It is shown that other CSAs do not fulfill all three properties.

In Sect. 3.3, two upper bounds on peak jitter achieved by LS-CSAs are presented and compared. The first bound meets a lower bound first presented in [LL84]; this bound does not describe the transient phase of synchronization, and there is a large gap between the bound and typically achieved peak-jitter values. The second bound is original work; it accurately describes the transient phase of synchronization and provides a better upper bound in the steady state than the first bound.

Section 3.4 discusses under what conditions it is possible to estimate and compensate the drift of the client's local clock. Constant and variable drift are considered. An LS-CSA is presented that implements drift compensation using a-priori information about message delays. Upper bounds on the error of drift compensation are derived. It is shown that drift compensation dramatically improves synchronization.

In Sect. 3.5, we present simple, implementable heuristic LS-CSAs that do drift compensation without requiring any a-priori information about message delays. It is discussed which of the properties of Sect.3.2 still hold and which do not. The heuristic LS-CSAs are compared in terms of the cost and performance achieved in a 802.11b WLAN.

In Sect. 3.6, the performance of the heuristic LS-CSAs is compared with the performance of other CSAs. To this purpose, the wireless-loudspeakers application is presented and a set of target performance metrics is defined that model the requirements of this application. The quality of synchronization is examined in different 802.11b WLAN scenarios. The performance of the CSAs is compared in various scenarios and network load situations.

3.1 Local-Selection Principle

In this section, the Basic Local-Selection algorithm $\mathcal{A}^{\text{ls}}$ is introduced and the class of Local-Selection CSAs is formally defined. The difference between Local-Selection CSAs and a similar algorithm from Lamport [Lam78] is discussed.

3.1.1 Definition and Description

The presentation starts with the definition of a very simple CSA, which we call Basic Local-Selection algorithm.

Def. 16: **(Basic Local Selection)** *The Basic Local-Selection algorithm $\mathcal{A}^{\text{ls}}$ is a selective estimate-based CSA (Def. 7) that computes candidate clocks C_i^* and decision π_i from the view $\mathcal{V}_i = \{(s_j, h_j) | j \leq i\}$ and the parameter $\hat{\rho}$ according to the following rules:*

$$C_i^*(h) = s_i + \frac{h - h_i}{1 + \hat{\rho}}, \quad \forall h \geq h_i \tag{3.1}$$

$$\pi_i = (c_i^* > c_i^-), \quad \forall i > 1 \tag{3.2}$$

The parameter $\hat{\rho}$ is an upper bound on the drift of the local clock.

An example of how the algorithms works is shown in Fig. 18. In the following, three properties are described that capture the essential behavior of this algorithm. These properties will then be used to define the *class* of Local Selection CSAs.

Prop. 1: **(Bounded Negative Error)** *The error of every candidate value c_i^* is negative or zero and finite.*

$$-\infty < e_i^* \leq 0 \;\; \forall i$$

Prop. 2: **(Slow Logical Clocks)** *The drift of all logical clocks is negative and lower-bounded by a constant $-\hat{\varrho}$.*

$$-\hat{\varrho} < \varrho_i(t) < 0 \;\; \forall i, \forall t \geq t_i$$

Prop. 3: **(Required Positive Adjustment)** *The synchronization error e_i immediately after starting a new logical clock is (i) not smaller than it was before (e_i^-) and (ii) not smaller than the candidate error e_i^*.*

$$e_i \geq \max(e_i^-, e_i^*) \;\; \forall i > 1$$

Based on these properties, we define the *class* of Local-Selection CSAs.

Def. 17: **(Local-Selection CSAs)** *A selective estimate-based CSA $\mathcal{A}$ is a Local Selection CSA (LS-CSA) if Properties 1, 2 and 3 hold.*

Lem. 1: *The Basic Local-Selection algorithm $\mathcal{A}^{\text{ls}}$ is an LS-CSA.*

Proof:

Prop. 1 The candidate error is $e_i^* = s_i - t_i$. Since s_i is the send time and t_i is the receive time of the i-th time-stamp message, the error is $e_i^* = -d_i$. Every message has a positive delay and thus $e_i^* < 0$. Every message that is received has a finite delay, and thus $-\infty < e_i^*$.

Prop. 2 The clock drift is $\varrho_i(t) = \frac{\mathrm{d}C_i(H(t))}{\mathrm{d}t} - 1 = \frac{1+\rho(t)}{1+\hat{\rho}} - 1$. By Ass. 1, $\rho(t) < \hat{\rho}$, and thus $\varrho_i(t) < 0$. Also by Ass. 1, $\rho(t) > -\hat{\rho}$, and thus $\varrho_i(t) > -\hat{\varrho} = \frac{1-\hat{\rho}}{1+\hat{\rho}} - 1 \approx -2\hat{\rho}$.

Prop. 3 *Case 1)* If $c_i^* > c_i^-$, then $c_i = c_i^* = s_i > c_i^-$. Subtracting t_i, we get $c_i - t_i = c_i^* - t_i > c_i^- - t_i$ and thus $e_i = e_i^* > e_i^-$. *Case 2)* If $c_i^* \leq c_i^-$, then $c_i = c_i^- \geq s_i = c_i^*$. Subtracting t_i results in $e_i = e_i^- \geq e_i^*$. □

3.1.2 Comparison with Lamport's CSA

In order to provide some intuitive understanding of the characteristics of LS-CSAs, we compare algorithm $\mathcal{A}^{\text{ls}}$ with an algorithm presented by Lamport in the seminal paper [Lam78].

Def. 18: **(Lamport's CSA)** *Algorithm $\mathcal{A}^{\text{lam}}$ is a selective estimate-based CSA that computes candidate clocks C_i^* and decisions π_i from the view $\mathcal{V}_i = \{(s_j, h_j) | j \leq i\}$ according to the following rules:*

$$C_i^*(h) = s_i + h - h_i, \quad \forall h \geq h_i \tag{3.3}$$

$$\pi_i = (c_i^* > c_i^-), \quad \forall i > 1 \tag{3.4}$$

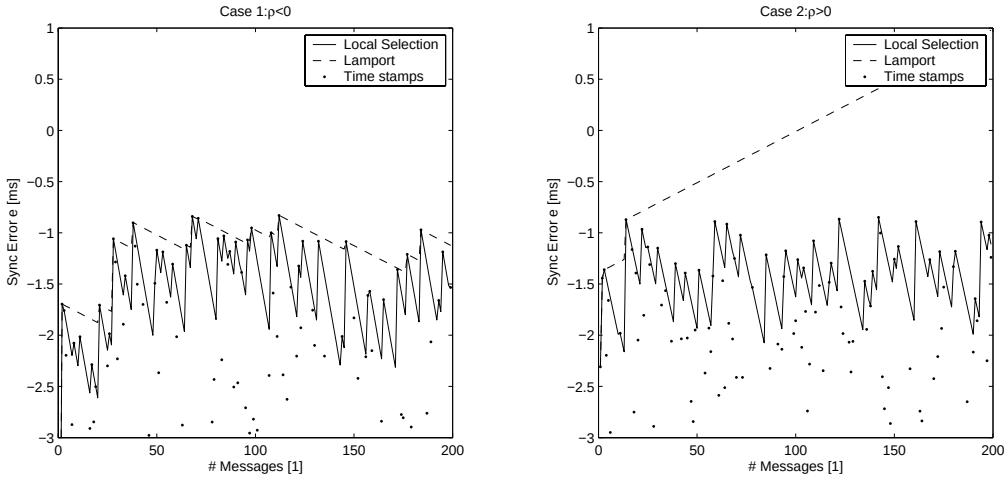


Fig. 18: *Synchronization error achieved by Lamport's algorithm and by the Basic Local-Selection algorithm.* On the left, the case of a slow client clock is shown. Both algorithms work correctly, the performance of the Local-Selection algorithm is worse because the absolute value of the client's drift is by $\hat{\rho}$ larger than in the case of Lamport's algorithm. On the right, the case of a fast client is displayed. Here, Lamport's algorithm fails to synchronize. The Local Selection algorithm works correctly as in the slow client case.

The algorithm $\mathcal{A}^{\mathrm{lam}}$ is similar to $\mathcal{A}^{\mathrm{ls}}$ in that it only makes positive adjustments to the synchronized clock C, and thus Prop. 3 holds. Moreover, every logical clock that is initially different from its predecessor has a negative error, thus Prop. 1 holds. But it is not an LS-CSA, since the drift of the logical clocks is $\varrho_i(t) = \rho(t)$, which can be either positive or negative. The consequences of this difference are illustrated in Fig. 18.

Case 1: $\rho(t) \leq 0$
If the client's local clock H progresses slower than real time, the client keeps updating its clock forever. It does not update its clock at every reception of a time stamp s_i, e.g. if a very slow message follows shortly after a fast message, it will be ignored. But after some time, the client's clock has fallen far enough behind real time that even a slow message is used to update the client's clock.

Case 2: $\rho(t) > 0$
If the client's local clock progresses faster than real time, the situation is different: In case the client's clock initially is behind real time, some of the first received messages are used to update the client's clock. But at some point of time, the client's clock gets ahead of real time and all messages received from this time on are discarded without updating the client's clock. The client's clock now freely runs away of real time, boundlessly increasing the synchronization error.

Discussion
We have shown that algorithm $\mathcal{A}^{\mathrm{lam}}$ fails to synchronize if the local clock is faster than the reference clock. In [Lam78], a slightly different setting is described, in which this problem is avoided: Lamport's algorithm synchronizes a number of nodes with drifting clocks among each other. *Every node* periodically sends time-stamp messages to all other nodes and whenever a node receives a time stamp with a higher value than its own time, it sets its current time to the value of the received time stamp. Thus the node with the fastest clock adjusts its clock only upon receiving a time stamp from the node with the highest offset and then stops updating for good. All other nodes keep adjusting their clocks to the time stamps from the node with the fastest clock. In this setting, the "reference node" is not specified a priori, but emerges during the execution of the algorithm and may also change when the drift rates of the local clocks vary. This synchronization scheme is part of the IEEE 802.11 standards, where it is called *Distributed Time Synchronization Function (DTSF)*. The scheme has two drawbacks: (i) If the number of nodes sending time-stamp messages is large, there may result congestion and in consequence loss of messages. If so, the node with the fastest local clock may not be able to send time-stamp messages to some other nodes, and thus synchronization fails. This scalability problem has been described in [HL02]. (ii) Relating the synchronized time to some external time (e.g. UTC) is non-trivial, since it cannot be predicted which node takes the role of the reference. These system-wide synchronization issues will be discussed in more detail in Chaps. 4 and 5.

Returning to the issue of this chapter, namely the synchronization of a client node with a predetermined reference node, the LS-CSAs have the advantage over algorithm $\mathcal{A}^{\mathrm{lam}}$ that they always operate in the mode described as case 1 and thus avoid free-running synchronized clocks and unbounded synchronization error. This claim is formalized as the liveness property in Sect. 3.2. On the other hand, the left side of Fig. 18 shows that peak jitter and the other performance metrics achieved by $\mathcal{A}^{\mathrm{ls}}$ are worse than those achieved by $\mathcal{A}^{\mathrm{lam}}$ if the local clock H is already slower than real time. This is discussed in Sect. 3.3, where upper bounds on the performance metrics are derived. In Sects. 3.4 and 3.5, Local-Selection algorithms and heuristics are presented that eliminate this drawback by drift compensation, i.e. by making sure that $\varrho_i(t)$ is negative, but only *as little as necessary*.

3.2 Properties

In this section, we show that Local-Selection CSAs are (i) safe and live and (ii) that they are optimally selective. These properties formalize *correct* behavior of a CSA. A safe CSA never actively degrades synchronization, and a live CSA *eventually* improves synchronization. Optimal selectivity is stronger in the sense that it implies safety and requires that a CSA improves synchronization

whenever possible. This section is summarized in the following theorem:

Thm. 2: *Every LS-CSA is safe, live, and optimally selective.*

First we prove these three properties and then compare the properties of LS-CSAs with the properties of other CSAs.

3.2.1 Properties of Local-Selection CSAs

Before proving the properties safety, liveness and optimal selectivity, we start with the following lemma:

Lem. 3: **(Negative Offset)** *The error E of a synchronized clock computed by an LS-CSA is negative or zero.*

$$E(t) \leq 0 \;\; \forall t > t_1$$

Proof: Every logical clock C_i that is different from its predecessor starts with a negative error $e_i = e_i^* \leq 0$. (Prop. 1). Since the drift of every logical clock is negative (Prop. 2), the error remains negative.

If all logical clocks that are not equal to the previous logical clock have negative error, then *all* logical clocks have negative error and then also the synchronization error is negative. □

Lemma 3 actually states that an LS-CSA computes a valid lower bound on real time. This will be of interest in the next chapter, where we discuss interval-based CSAs. The lemma is presented here, because it shortens the proof of optimal selectivity.

Lem. 4: **(Safety)** *Every LS-CSA is safe.*

Proof: *Case 1)* If a logical clock is equal to its predecessor, then $e_i = e_i^-$, fulfilling the safety condition $|e_i| \leq |e_i^-|$.

Case 2) By Prop. 1, the initial error of every logical clock that is different from its predecessor is negative, i.e. $e_i = e_i^* \leq 0$. By Prop. 3, we have $e_i \geq e_i^-$. Combining these relations, we get $0 \geq e_i \geq e_i^-$ and thus $|e_i| \leq |e_i^-|$, which corresponds to Def. 8. □

Lem. 5: **(Liveness)** *Every LS-CSA is live.*

Proof: To prove liveness, we have to show that the synchronization error eventually improves (Def. 9). Since we have already shown safety, it suffices to show that eventually, a logical clock C_i is different from its predecessor.

Let $e^* = \min_i(e_i^*) > -\infty$ (by Prop. 1). Any logical clock C_i eventually has an error that is smaller than e^* (by Prop. 2). Let t_k be the real time of the first message arrival after the error of C_i has fallen below e^*. Assume that no logical clock C_j with $i < j < k$ is different from C_i, then $e_k^- < e^*$. By Prop. 3, we have $e_k \geq e_k^* \geq e^*$ and $e_k > e_k^-$, thus C_k is different from its predecessor. □

Lem. 6: (Optimal Selectivity) *Every LS-CSA is optimally selective.*

Proof: *Step 1)* If $|e_i^*| > |e_i^-|$, then $e_i^* < e_i^-$ (Prop. 1 and Lem. 3). Adding t_i to both sides, we get $c_i^* = s_i < c_i^-$ and thus $\pi_i = (s_i > c_i^-) = 0$.

Step 2) If $|e_i^*| < |e_i^-|$, then $e_i^* > e_i^-$ (Prop. 1 and Lem. 3). Adding t_i to both sides, we get $c_i^* = s_i > c_i^-$ and thus $\pi_i = (s_i > c_i^-) = 1$. □

3.2.2 Discussion

First, we discuss the validity of the presented properties for non-LS-CSAs, and then summarize Sect. 3.2

Algorithm $\mathcal{A}^{\text{lam}}$ is safe, but not live and not optimally selective. Safety can be shown in the same way as for LS-CSAs, as the proof of Lem. 4 requires only Props. 1 and 3, which also hold for $\mathcal{A}^{\text{lam}}$. Algorithm $\mathcal{A}^{\text{lam}}$ is neither live nor optimally selective: A counterexample is shown in Fig. 18. This is because $\mathcal{A}^{\text{lam}}$ does not fulfill Prop. 2, and thus it cannot be shown that $e_i^- < 0, \forall i > 1$.

It can easily be shown that $\mathcal{A}^{\text{pll}}$ and $\mathcal{A}^{\text{llr}}$ presented in the previous chapter are not safe and thus not optimally selective. The reason is that every received time stamp influences new logical clocks, regardless of the accuracy of the time stamp. E.g. on a delay sequence with $d_i = 0$ for $i \in [1, 9]$ and $d_{10} = 1\text{s}$, the error of synchronized clocks computed by these CSAs increases at the reception of the tenth time-stamp message, thus these CSAs are not safe. These algorithms are live on most delay sequences. The outlier-rejection mechanism plays an important role: a small threshold value leads to rejection of most time stamps, thus the algorithms tend to be "more safe, but less live". Hard statements are very difficult to prove.

Cristian's algorithm presented in [Cri89] is not a unidirectional CSA. In contrast to all other CSAs presented here it requires a view $\mathcal{V}_i$ with round-trip measurements, i.e. the client asks for time stamps from the reference and measures the time between sending the request and receiving the time stamp. Cristian's CSA is considered here because it is a truly selective estimate-based CSA. A logical clock is different from its predecessor if and only if the measured round-trip delay is below a specified threshold value. But this condition is neither related to the synchronization error immediately before the update, nor to the synchronization error after the update. Even if the round-trip delay is below the threshold value, there is no guarantee that the candidate value c_i^* computed under the assumption of symmetrical delays from and to the reference node does not provide worse accuracy than the current synchronized time c_i^-. On the other hand, even if the round-trip delay is large, the candidate value c_i^* can be perfectly accurate ($e_i^* = 0$), provided that the delays from and to the reference node are symmetric. The algorithm is thus neither safe nor optimally selective.

In Tab. 6, the validity of the various properties presented in this section are summarized for the CSAs presented so far and also for the CSAs presented in the following sections. Only the Local-Selection algorithm satisfies safety, liveness and optimal selectivity. The other algorithms either can degenerate into

Algorithm		*Safety*	*Liveness*	*Opt. Selectivity*	*Peak Jitter*
$\mathcal{A}^{\text{loc}}$		•	-	-	∞
$\mathcal{A}^{\text{net}}$		-	-	-	J_s
$\mathcal{A}^{\text{llr}}$		(-)	(•)	(-)	?
$\mathcal{A}^{\text{pll}}$		(-)	(•)	(-)	?
$\mathcal{A}$ from [Cri89]		(-)	(•)	(-)	?
$\mathcal{A}^{\text{lam}}$ ($\rho > 0$)		•	-	-	∞
$\mathcal{A}^{\text{lam}}$ ($\rho \leq 0$)		•	•	•	J_s, J_d
LS	$\mathcal{A}^{\text{ls}}$	•	•	•	
	$\mathcal{A}^{\text{lsdel}}$	if $D_i < d_i$	•	if $D_i < d_i$	
	$\mathcal{A}^{\text{lsdc}}$	•	if $R_i(H(t)) > \rho(t),\ \forall t > t_i$		
	$\mathcal{A}^{\text{lsdc,iter}}$		if D^u is valid		J_s, J_d, J_d^l
HLS	$\mathcal{A}^{\text{leak}}$	•	if $\lambda > \frac{\hat{\vartheta}}{2(1-\hat{\rho})}$	-	?

Tab. 6: *Properties of various CSAs.* The symbol • implies that the property holds, − means that it does not. $(\bullet/-)$ marks unproved conjectures. Local-Selection (LS) CSAs fulfill all properties (Sect. 3.2); peak jitter J is bounded by J_s and J_d (Sect. 3.3). The peak jitter of algorithms that are not live cannot be bounded. For some algorithms, e.g. Heuristic Local-Selection (HLS) CSAs (Sect. 3.5), bounds on peak jitter are not known.

a mode where they stop doing anything and thus degrade the accuracy boundlessly (not live), or may actively degrade the accuracy by their actions (not safe).

The presented properties are a novel way to analyze the behavior of CSAs on a *microscopic level*, i.e. considering their reaction to the input of a single time-stamp message. The advantage of this approach is that the analysis is mostly independent of properties of system traces (except the exclusion of degenerated delay sequences for the liveness property of $\mathcal{A}^{\text{ls}}$, see Def. 9). However, the presented properties cannot be used to draw conclusions about the average or worst-case accuracy, peak jitter, and MTIE of CSAs. Still, the optimal-selectivity property will be used to derive an upper bound on the peak jitter in the next section.

3.3 Upper Bounds on Performance Metrics

In this section, two different *upper bounds on the peak jitter* J_s and J_d achieved by Local-Selection CSAs are presented. The difference between the bounds is due to different adversaries that generate a delay sequence for time-stamp messages processed by LS-CSAs. Bound J_s is derived under the assumption that an adversary chooses every message delay d_i deliberately from the interval $[d_{\min}, d_{\max}]$. The bound J_d is derived under the assumption that the message delays d_i are constrained by an upper delay-interval curve D^u. In comparison, the bound J_d has a number of advantages:

- For realistic system parameters, J_d is far smaller than J_s.
- In real systems, the maximal delay $d_{\max}$ tends to be very large, due to sporadic outliers. J_s directly increases with $d_{\max}$, while the new bound J_d is mostly independent of the maximal delay.
- The bound J_d is a function of the setup time. It is intuitively clear that the peak jitter immediately after starting some algorithm $\mathcal{A}$ is large and later becomes smaller. Bound J_d derived from the delay-interval curve accurately describes this behavior, while J_s is independent of the setup time.

We first derive the two bounds on peak jitter and illustrate the difference in Ex. 9 and Fig. 19. Then the relation between bounds on peak jitter J and bounds on the accuracy A and the maximum time-interval error M are discussed.

3.3.1 Adversary 1: Bounded Delay

In the following we derive a bound on the peak jitter J achieved by an LS-CSA. The bound J_s is in accordance with the lower bounds presented in [LL84] for non-drifting nodes and in [HMM85] for nodes with bounded drift.

Thm. 7: (Static Upper Bound on Peak Jitter) *The peak jitter J achieved by algorithm $\mathcal{A}^{\text{ls}}$ is upper-bounded by $J_s \geq J$ with*

$$J_s = d_{\max} - d_{\min} + \hat{\varrho}\Delta t$$

if the delay d_i of every time-stamp message is contained in $[d_{\min}, d_{\max}]$ and the interval between consecutive message arrivals is at most Δt.

Proof: By definition, we have $J = \max(E(t)) - \min(E(t))$. By Prop. 2, the error of logical clocks only decreases, thus $\max(E(t)) = \max(e_i^*) = \max(-d_i) \leq -d_{\min}$. Let $E(t') = \min(E(t))$. If we can show that $E(t') \geq -d_{\max} - \hat{\varrho}\Delta t$, then $J \leq -d_{\min} + d_{\max} + \hat{\varrho}\Delta t = J_s$.

The rest of the proof is by contradiction: Assume that $E(t') < -d_{\max} - \hat{\varrho}\Delta t$. By Ass. 3, we know that at some time $t_i \in [t' - \Delta t, t']$ a time-stamp message with a delay $d \leq d_{\max}$ has been received.

Case 1) If at this message arrival (real time t_i) the synchronized clock has been set to the value of the time stamp s_i, then the synchronization error is $E(t') \geq -d_{\max} + \int_{t_i}^{t'} \varrho_i(t)\mathrm{d}t$. Since $\varrho(t) > -\hat{\varrho}$, we get $E(t') \geq -d_{\max} - \hat{\varrho}\Delta t$, which contradicts the assumption.

Case 2) If at t_i the synchronized clock has not been set to s_i, then $s_i = t_i - d_i$ must have been smaller than $C(t_i)$. Subtracting t_i, we get $-d_i < E(t_i)$. As $E(t_i) = E(t') - \int_{t_i}^{t'} \varrho_i(t)\mathrm{d}t$ and $\varrho_i(t) > -\hat{\varrho}$, we derive $E(t_i) < E(t') + \hat{\varrho}(t' - t_i)$. Using the original assumption, we arrive at $E(t_i) < -d_{\max}$. But $-d_i < E(t_i) < -d_{\max}$ is a contradiction. □

The peak jitter thus depends on the maximal and the minimal message delay $d_{\max}$ and $d_{\min}$, the maximal clock drift $\hat{\varrho}$, and the interval Δt between time-stamp messages. As $\hat{\varrho}$ typically is very small ($\leq 100\text{ppm} = 10^{-4}$), the delay component ($d_{\max} - d_{\min}$) most often dominates over the drift component ($\hat{\varrho}\Delta t$).

If the maximal delay is very large, then also the bound on peak jitter becomes very bad. Unfortunately, in most real situations, *the maximal delay is indeed very large*, compared to typical (median) delay values, see e.g. Tab. 3. In these situations, the derived bound is of little use if the performance of a CSA has to be estimated realistically; the bound is far too pessimistic.

3.3.2 Adversary 2: Upper Delay-Interval Curve

We now present the upper bound J_d on the peak jitter achieved by an LS-CSA.

Thm. 8: (Dynamic Upper Bound on Peak Jitter) *The peak jitter J achieved by algorithm $\mathcal{A}^{\text{ls}}$ after a setup time of $i\Delta t$ is upper-bounded by $J_d(i\Delta t) \geq J$ with*

$$J_d(i\Delta t) = \min\{D^u(j) + \hat{\varrho} j\Delta t | j \in [1, i]\} - D^u(\infty)$$

if the delays d_i of the time-stamp messages are constrained by the upper delay-interval curve D^u and the interval between consecutive message arrivals is at most Δt.

Note that the bound J_s from Thm. 7 is a special case of the bound J_d from Thm. 8: $J_s = J_d(\Delta t) = D^u(1) - D^u(\infty) + \hat{\varrho}\Delta t$. By Ass. 5, $D^u(1) = d_{\max}$ and $D^u(\infty) = d_{\min}$, and thus $J_d(\Delta t) = d_{\max} - d_{\min} + \hat{\varrho}\Delta t$.

Proof: The bound J_s is based on the observation that whenever a time-stamp message is received, the error cannot be smaller than $-d_{\max}$, and between two messages, the error cannot decrease by more than $\hat{\varrho}\Delta t$.

Knowing that the message delays are constrained by D^u allows to extend this argument: $D^u(i)$ implies that in every interval of length $i\Delta t$, at least one message is received with a delay $d \leq D^u(i)$. At these moments, the error cannot be smaller than $-D^u(i)$, and between these moments, the error cannot decrease by more than $-\hat{\varrho} i\Delta t$. The first of these messages with $d \leq D^u(j)$ arrives at latest $i\Delta t$ after the first time-stamp message. Thus, the peak jitter J achieved after a setup time of $i\Delta t$ is at most $D^u(i) - D^u(\infty) + \hat{\varrho} i\Delta t\}$. The same argument can be made for any $1 \leq j \leq i$, and therefore $J_d(i\Delta t) = \min\{D^u(j) + \hat{\varrho} j\Delta t | j \in [1, .., i]\} - D^u(\infty)$. □

Ex. 9: *To illustrate Thm. 8, we generate 50 delay sequences with $d_{\max} = 5\text{ms}$, $d_{\min} = 0\text{ms}$ and a message interval $\Delta t = 0.1\text{s}$. Additionally, the delay sequences are generated such that they comply with the upper delay-interval curve $D^u(i) = (\frac{51-i}{50})^5 d_{\max}$ for $i \leq 50$ and $D^u(i) = (\frac{1}{50})^5 d_{\max}$ for $i > 50$. The Local-Selection algorithm is simulated with these delay sequences, assuming a node with $\rho = -\hat{\rho} = -100\text{ppm}$. The resulting peak jitter in function of the setup time is shown in Fig. 19.*

Clearly, the static bound $J_s = d_{\max} - d_{\min} + 2\hat{\rho}\Delta t = 5.02\text{ms}$ (dashed line in Fig. 19) is a valid upper bound for all the traces. While the bound is quite accurate at the start of the sequence, the peak jitter soon becomes far smaller than can be predicted using this bound, e.g. the peak jitter is below $630\mu\text{s}$ on all delay sequences after 2s.

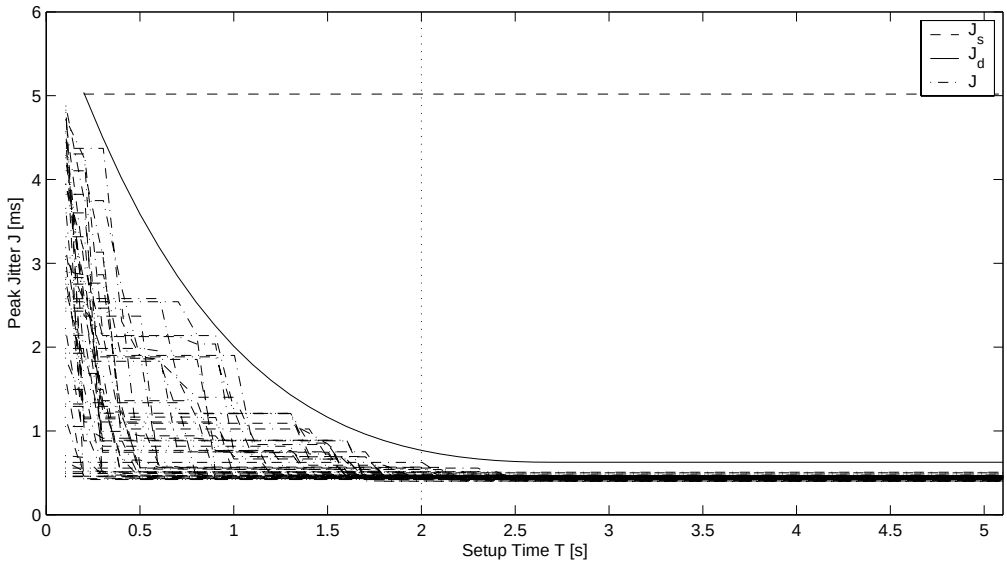


Fig. 19: *Upper bounds on peak jitter.* The dashed line represents bound J_s (Thm. 7). The solid line shows bound J_d (Thm. 8). The dash-dotted lines represent the peak jitter J in function of the setup time achieved by the Basic Local-Selection algorithm on 50 delay sequences. Both bounds are valid, but J_d is much tighter. For example after $2\mathrm{s}$ (dotted line), $J_s = 5.02\mathrm{ms}$, $J_d = 770\mu\mathrm{s}$ and $J < 630\mu\mathrm{s}$. Bound J_d also describes the behavior in the transient phase at the start of the delay sequences.

The dynamic bound J_d is shown as a solid line in Fig. 19. It can be seen that it is also valid for all the traces, but describes much more accurately the actual peak jitter achieved by the Local-Selection algorithm. For example, after $2\mathrm{s}$ setup time, a peak jitter of at most $770\mu\mathrm{s}$ is predicted.

Discussion

The dynamic bound J_d provides better estimates of the peak jitter achieved by algorithm $\mathcal{A}^{\mathrm{ls}}$ than the static bound J_s: (i) The bound is dynamic, i.e. the bound decreases for increasing setup time. The bound thus reflects the transient behavior at the begin of a time-stamp-message sequence. (ii) The new bounds provide much better guarantees. In the numeric example shown in Fig. 19, the upper bound on peak jitter is reduced from $5\mathrm{ms}$ to $770\mu\mathrm{s}$, whereas $\mathcal{A}^{\mathrm{ls}}$ achieves peak jitter of at most $630\mu\mathrm{s}$ on 50 sequences.

This benefit is due to the additional delay constraint, the upper delay-interval curve D^u. On the one hand, such a curve D^u is a stronger requirement than simple upper and lower bounds $d_{\min}$ and $d_{\max}$ (explaining intuitively the better bound). On the other hand, it could also be argued that D^u is a *weaker* requirement, since $D^u(1) \to \infty$ has not much significance for J_d, whereas $d_{\max} \to \infty$ makes J_s utterly worthless. The bound J_d is robust in the sense that it does not depend on $d_{\max}$, which in practical settings often is much larger than most of the delays.

Comparison to Lamport's Algorithm
No finite bound can be derived for algorithm $\mathcal{A}^{\text{lam}}$ if the client's local clock is faster than real time. Otherwise, i.e. if $\rho < 0$, the bounds of Thms. 7 and 8 are also valid for algorithm $\mathcal{A}^{\text{lam}}$. Here we have $\hat{\varrho} = \hat{\rho}$ (for $\mathcal{A}^{\text{ls}}$, we had $\hat{\varrho} = 2\hat{\rho}$), and thus the peak jitter is better than in the case of algorithm $\mathcal{A}^{\text{ls}}$.

3.3.3 Upper Bounds on Accuracy

By Defs. 11 and 12, the peak jitter J and the accuracy A are related by

$$A = \begin{cases} J + \min(E(t)) & \text{if } \max(E(t)) \geq -\min(E(t)) \\ J - \max(E(t)) & \text{otherwise} \end{cases} \tag{3.5}$$

For algorithm $\mathcal{A}^{\text{ls}}$, the maximal error is bounded by $\max(E(t)) \leq -d_{\min}$ and $\max(E(t)) < -\min(E(t))$, thus we have $A \geq J + d_{\min}$. In the following, we discuss whether an LS-CSA can be constructed that produces a better accuracy A. Consider the following CSA:

Def. 19: (Local Selection with Delay Compensation) *The Local Selection with Delay Compensation algorithm $\mathcal{A}^{\text{lsdel}}$ is a selective estimate-based CSA that computes candidate logical clocks C_i^* from the view $\mathcal{V}_i = \{(s_j, h_j) | j \leq i\}$ according to the following rules:*

$$C_i^*(h) = s_i + D_i + \frac{h - h_i}{1 + \hat{\rho}}, \quad \forall h \geq h_i \tag{3.6}$$

$$\pi_i = (c_i^* > c_i^-), \quad \forall i > 1 \tag{3.7}$$

The constant D_i is either an a-priori specified parameter or computed from the view $\mathcal{V}_i$.

Lem. 9: (Local Selection with Delay Compensation) *Algorithm $\mathcal{A}^{\text{lsdel}}$ is an LS-CSA, if $D_i \leq d_i, \ \forall i$. Then the accuracy is $A \geq J + d_{\min} - D_i$.*

Proof: The difference between algorithms $\mathcal{A}^{\text{lsdel}}$ and $\mathcal{A}^{\text{ls}}$ may cause a violation of Prop. 1, i.e. $e_i^* = s_i + D_i - t_i = D_i - d_i$ may be positive. This is not the case if $D_i \leq d_i$. □

If $D_i > d_i$, then algorithm $\mathcal{A}^{\text{lsdel}}$ is not safe, it is not optimally selective and Lem. 3 does not hold. However, algorithm $\mathcal{A}^{\text{lsdel}}$ is still live, since this only requires $-\infty < e_i^*$ and negative drift of the logical clocks. Also the upper bounds on peak jitter remain valid.

How to determine D_i
The problem of finding D_i can be interpreted as *delay compensation* and $D_i = d_{\min}$ obviously is the maximal D_i that guarantees safety and optimal selectivity. In this case, we have $A = J$. As will be shown in Sect. 3.5, it is not possible to estimate $d_{\min}$ if no a-priori knowledge about message delays is provided. We will return to this problem of estimating the minimal delay at the end of Chap. 4.

As a consequence, the accuracy will be neglected in the next sections in favor of peak jitter[1].

3.3.4 Upper Bound on MTIE

In this section we have provided upper bounds on the peak jitter and the accuracy provided by LS-CSAs. There is a trivial relation between peak jitter and MTIE: $M_\tau \leq J \ \ \forall \tau$, since $J = M_\infty$ and M is monotonic increasing with the window size τ. It is not clear whether a better bound on the MTIE can be found - we conjecture that under the two delay models used here, no tighter bound can be found. The reason is that there is no constraint on the occurrence of messages with minimal delay, and therefore $\max(E(t)) \leq d_{\min} + D_i$ seems to be the only bound for $\max(E(t))$, no matter how small or large the interval length τ is.

3.4 Drift Compensation

In this section, we present methods to estimate the drift ρ of the local clock under the assumption that (i) ρ is constant (Ass. 1 and $\vartheta = 0$) and (ii) the drift variation ϑ is bounded (Ass. 2). An LS-CSA is presented that employs these methods. An upper bound on the achieved peak jitter is derived, which shows that drift compensation dramatically improves the synchronization quality, but requires a relatively long setup time.

Def. 20: (Local Selection with Drift Compensation) *The Local-Selection with Drift Compensation algorithm $\mathcal{A}^{\text{lsdc}}$ is a selective estimate-based CSA that computes the candidate logical clock C_i^* and decision π_i from the view $\mathcal{V}_i = \{(s_j, h_j) | j \leq i\}$ according to the following rules:*

$$C_i^*(h) = s_i + \frac{h - h_i}{1 + R_i(h)}, \quad \forall h \geq h_i \tag{3.8}$$

$$\pi_i = (c_i^* > c_i^-), \quad \forall i > 1 \tag{3.9}$$

The function R_i is a parameter either a priori specified or computed from the view $\mathcal{V}_i$.

Thm. 10: *Algorithm $\mathcal{A}^{\text{lsdc}}$ is an LS-CSA if $R_i(H(t)) > \rho(t), \ \forall t > t_i$.*

Proof: Of the three defining properties of an LS-CSA, only Prop. 2 (slow logical clocks) is affected by the difference between $\mathcal{A}^{\text{ls}}$ and $\mathcal{A}^{\text{lsdc}}$. Thus, if $\varrho_i(t) = \frac{1+\rho(t)}{1+R_i(H(t))} - 1 < 0$, then algorithm $\mathcal{A}^{\text{ls}}$ is an LS-CSA. Clearly, Prop. 2 holds if $R_i(H(t)) > \rho(t) \ \forall t > t_i$. □

[1] A final remark concerning the accuracy: If minimizing accuracy A is the only goal and the properties safety and optimal selectivity are not of concern, then $D_i = d_{\min} + J/2$ is the best choice. In this case, we get $A = J/2$.

If Prop. 2 does not hold, algorithm $\mathcal{A}^{\mathrm{lsdc}}$ is neither live nor optimally selective, exactly as $\mathcal{A}^{\mathrm{lam}}$ on a node with a fast local clock. Safety still holds, regardless of the drift of the local clock.

The problem of finding R_i can be interpreted as *drift compensation*. Clearly, if no assumptions are made about the variation $\vartheta = \frac{\mathrm{d}\rho}{\mathrm{d}t}$ of the local clock drift, then no $R_i(H(t)) < \hat{\rho}$ can guarantee $R_i(H(t)) > \rho(t)$.

3.4.1 Constant Drift

We now present a method to estimate the drift ρ of the local clock under the assumption that ρ is constant. The method computes the constant function R_i, which is an upper bound on the drift ρ.

Thm. 11: ***(Estimation of Constant Drift)*** *Let $h_j < h_i$ be the local times of arbitrary time-stamp-message arrivals, let c_j and c_i be the synchronized times associated with these events, and let J be an upper bound on the peak jitter in the real-time interval $[t_j, t_i]$. Then the constant R_i computed according to the rule*

$$R_i = \frac{h_i - h_j}{c_i - c_j - J} - 1 \tag{3.10}$$

is an upper bound on the drift of the local clock, i.e. $\rho \leq R_i$. The drift ϱ_i of the logical clock computed according to Def. 20 is lower-bounded by $-\hat{\varrho} \leq \varrho_i$, with

$$\hat{\varrho} = \frac{2J}{c_i - c_j + J} \tag{3.11}$$

Proof: As we assume constant ρ, the relation $\rho = \frac{h_i - h_j}{t_i - t_j} - 1$ holds for all $j < i$. The real-time difference $t_i - t_j$ is not known by a CSA, but it can be bounded by $c_i - c_j + J \geq t_i - t_j \geq c_i - c_j - J$, as J is an upper bound on the peak jitter in the whole real-time interval $[t_j, t_i]$. Thus the drift can be bounded by $\frac{h_i - h_j}{c_i - c_j + J} - 1 \leq \rho \leq \frac{h_i - h_j}{c_i - c_j - J} - 1$. Therefore, $R_i = \frac{h_i - h_j}{c_i - c_j - J} - 1 \geq \rho$.

The drift of the logical clock is $\varrho_i = \frac{1+\rho}{1+R_i} - 1$. Using the bounds on the drift of the local clock, we get $\varrho_i \geq \frac{h_i - h_j}{c_i - c_j + J} / \frac{h_i - h_j}{c_i - c_j - J} - 1 = \frac{c_i - c_j - J}{c_i - c_j + J} - \frac{c_i - c_j + J}{c_i - c_j + J} = \frac{-2J}{c_i - c_j + J}$. Therefore, $-\hat{\varrho} \leq \varrho_i$ with $\hat{\varrho} = \frac{2J}{c_i - c_j + J}$. □

From Eq. 3.11, we see that the lower bound $\hat{\varrho}$ gets arbitrarily close to zero when the interval $[c_j, c_i]$ grows. The interpretation of this observation is that drift compensation with an arbitrarily small error is possible after a sufficiently long setup time.

3.4.2 Variable Drift

In practice, it is not possible to manufacture an oscillator device that is perfectly stable, and thus constant drift is often not a realistic assumption. We now consider the case that the variation of the drift $\vartheta(t) = \frac{\mathrm{d}\rho(t)}{\mathrm{d}t}$ is bounded by a known constant $\hat{\vartheta}$ (Ass. 2). In the following, it is shown that with variable drift, the error of drift compensation cannot be reduced arbitrarily.

We present a method that computes the constant r_i, which is an upper bound on $\rho(h_i)$, and the function R_i, which is an upper bound on $\rho(h),\ \forall h \geq h_i$.

Thm. 12: (Estimation of Variable Drift) *Let $h_j < h_i$ be the local time of arbitrary time-stamp-message arrivals, let c_j and c_i be the synchronized times associated with these events, and let J be an upper bound on the peak jitter in the interval $[t_j, t_i]$. Then the constant r_i computed according to the rule*

$$r_i = \frac{h_i - h_j}{c_i - c_j - J} + \frac{1}{2}\hat{\vartheta}(c_i - c_j + J) - 1 \tag{3.12}$$

is an upper bound on the drift of the local clock at real time t_i, i.e. $\rho(t_i) < r_i$. The function R_i computed as

$$R_i(h) = r_i + \hat{\vartheta}(C(h) - c_i + J) \tag{3.13}$$

is an upper bound on $\rho(t)$ for every $h = H(t) \geq h_i$.

The drift ϱ_i of the logical clock computed according to Def. 20 is approximately lower-bounded by $-\hat{\varrho} \leq \varrho_i(t_i)$, with

$$\hat{\varrho} = \frac{2J}{c_i - c_j - J} + \hat{\vartheta}(c_i - c_j + J) \tag{3.14}$$

Proof: The local-time difference is $h_i - h_j = t_i - t_j + \int_{t_j}^{t_i} \rho(t)\mathrm{d}t$. The clock drift at time t_i is maximal if it increases as much as possible over the whole interval $[t_j, t_i]$, i.e. $\rho(t) = \rho(t_j) + \hat{\vartheta}(t - t_j)$. Then $h_i - h_j = t_i - t_j + \int_{t_j}^{t_i} \Big(\rho(t_j) + \hat{\vartheta}(t - t_j)\Big)\,\mathrm{d}t = (1 + \rho(t_j))(t_i - t_j) + \frac{1}{2}\hat{\vartheta}(t_i - t_j)^2$. Resolving the equation for $\rho(t_j)$ results in $\rho(t_j) = \frac{h_i - h_j}{t_i - t_j} - \frac{1}{2}\hat{\vartheta}(t_i - t_j) - 1$. From the assumption of constantly increasing drift, we get $\rho(t_i) = \rho(t_j) + \hat{\vartheta}(t_i - t_j) = \frac{h_i - h_j}{t_i - t_j} + \frac{1}{2}\hat{\vartheta}(t_i - t_j) - 1$. As the real-time difference $t_i - t_j$ is bounded by $c_i - c_j - J \leq t_i - t_j \leq c_i - c_j + J$, we derive $\rho(t_i) \leq \frac{h_i - h_j}{c_i - c_j - J} + \frac{1}{2}\hat{\vartheta}(c_i - c_j + J) - 1 = r_i$.

If $r_i \geq \rho(t_i)$, then $r_i + \hat{\vartheta}(t - t_i) \geq \rho(t)$ for all $t \geq t_i$. As $t - t_i \leq C(H(t)) - c_i + J$, we have $R_i(H(t)) = r_i + \hat{\vartheta}(C(H(t)) - c_i + J) \geq \rho(t)$.

For the proof of $-\hat{\varrho} \leq \varrho_i(t_i)$, we approximate $\varrho_i(t_i) = \frac{1+\rho(t_i)}{1+r_i} - 1 \approx \rho(t_i) - r_i$, and obtain $\varrho_i(t_i) \geq \frac{h_i - h_j}{c_i - c_j + J} - \frac{1}{2}\hat{\vartheta}(c_i - c_j + J) - 1 - (\frac{h_i - h_j}{c_i - c_j - J} + \frac{1}{2}\hat{\vartheta}(c_i - c_j + J) - 1) = \frac{h_i - h_j}{c_i - c_j + J} - \frac{h_i - h_j}{c_i - c_j - J} - \hat{\vartheta}(c_i - c_j + J) = -\frac{2J(h_i - h_j)}{(c_i - c_j)^2 - J^2} - \hat{\vartheta}(c_i - c_j + J)$. With $h_i - h_j \leq (1 + \hat{\rho})(c_i - c_j + J)$, we get $\varrho_i(t_i) \geq -\frac{2J(1+\hat{\rho})}{c_i - c_j - J} - \hat{\vartheta}(c_i - c_j + J) \approx -\frac{2J}{c_i - c_j - J} - \hat{\vartheta}(c_i - c_j + J) = -\hat{\varrho}$.

□

Discussion
In contrast to the case of constant drift, the error of the drift estimation $\hat{\varrho}$ does not go to zero when the interval $[c_j, c_i]$ grows large. There is an optimal interval length, over which $\hat{\varrho}$ becomes minimal. Simplifying Eq. 3.14 to $\hat{\varrho} \approx \frac{2J}{c_i - c_j} + \hat{\vartheta}(c_i - c_j)$ results in a useful rule of thumb: The error of drift compensation attains the minimum

$$\hat{\varrho} \approx \sqrt{8J\hat{\vartheta}} \tag{3.15}$$

after a setup time of

$$c_i - c_j \approx \sqrt{\frac{2J}{\hat{\vartheta}}} \tag{3.16}$$

Ex. 10: *Assume a synchronized clock that has a peak jitter of at most* $J = 1\mathrm{ms}$*. If the oscillator stability is* $\hat{\vartheta} = 10^{-7}\mathrm{s}^{-1}$ *(*$0.1\mathrm{ppm/s}$*, typical for oscillators operated under strongly variable temperature conditions), the drift can be compensated with an error of* $30\mathrm{ppm}$ *after a setup time of* $140\mathrm{s}$*. If the oscillator stability is* $\hat{\vartheta} = 10^{-9}\mathrm{s}^{-1}$ *(*$3.6\mathrm{ppm/h}$*, typical for workstations in the first hour after power-up), the drift can be compensated with an error of* $3\mathrm{ppm}$ *after a setup time of* $1410\mathrm{s}$*. If the oscillator stability is* $\hat{\vartheta} = 10^{-11}\mathrm{s}^{-1}$ *(*$0.9\mathrm{ppm/day}$*, typical for long-running workstations in air-conditioned offices), the drift can be compensated with an error of* $0.3\mathrm{ppm}$ *after a setup time of* $14140\mathrm{s}$*.*

3.4.3 An expensive Local-Selection Variant

The drift-estimation methods presented in Sects. 3.4.1 and 3.4.2 both require that a bound J on the peak jitter is known. Thus a CSA that wants to employ these methods has to compute such a bound. This is possible, e.g. J_s can be used if the CSA is provided with upper and lower bounds on message delays, or J_d can be used if the CSA is provided with an upper delay-interval curve D^u.

In the following, we present the Local Selection with Iterative Drift Compensation algorithm $\mathcal{A}^{\mathrm{lsdc,iter}}$, shown in Alg. 4. The algorithm is designed for the assumption of constant drift ρ.

The algorithm has two parameters, the upper delay-interval curve D^u and the number of iterations L. In a first iteration, the simple $\mathcal{A}^{\mathrm{ls}}$ algorithm is performed (equivalent to $\mathcal{A}^{\mathrm{lsdc}}$ with $R_i = \hat{\rho}$). The bound J_d^l for the peak jitter of the synchronized clock C^l is computed from the a-priori known upper delay-interval curve D^u (according to Thm. 8). The synchronized clock C^l and the bound J_d^l are used to compute the estimate R_i^{l+1} of the drift of the local clock H and also the maximal error $\hat{\varrho}^{l+1}$ of this estimate. Now that we have better information about clock drift, the whole history of received messages is used to recompute a more accurate logical clock C^{l+1} and a smaller bound on peak jitter J_d^{l+1}. These results can then again provide a better estimation of the local clock drift and so forth.

Thm. 13: **(**$\mathcal{A}^{\mathrm{lsdc,iter}}$ **is an LS-CSA)** *Let the drift* ρ *of the local clock* H *be constant. Then the Local-Selection algorithm with Iterative Drift Compensation (Alg. 4) is an LS-CSA if* D^u *is a valid upper delay-interval curve.*

Algorithm 4 Local Selection with Iterative Drift Compensation $\mathcal{A}^{\text{lsdc,iter}}$

Parameters: Bound on clock drift $\hat{\rho}$, upper delay-interval curve D^u, number of iterations L
Input: View $\mathcal{V}_i = \{(s_j, h_j) | j \leq i\}$,
Output:
Logical clock C_i

for $l := 1$ **to** L **do**
  // Estimate receive interval Δt
  $\Delta t := (1 + \hat{\rho}) \max\{h_j - h_{j-1} | j \in [2, i]\}$

  // Compute R_i^l from synchronized clock C^l and bound J_d^l
  if $i = 1$ **then**
    $R_i^1 := \hat{\rho}$
  else
    $R_i^l := \min\left\{ \frac{h_i - h_j}{C^{l-1}(h_i) - C^{l-1}(h_j) - J_d^{l-1}(j\Delta t)} | j \in [1, i-1] \right\} - 1$ // (Thm. 11)
  end if

  // Compute $\hat{\varrho}^l$ from synchronized clock C^l and bound J_d^l
  if $i = 1$ **then**
    $\hat{\varrho}^1 := 2\hat{\rho}$
  else
    $\hat{\varrho}^l := \min\left\{ \frac{2J_d^{l-1}(j\Delta t)}{C^{l-1}(h_i) - C^{l-1}(h_j) + J_d^{l-1}(j\Delta t)} | j \in [1, i-1] \right\}$ // (Thm. 11)
  end if

  // Compute synchronized clock C^l with current drift compensation R_i^l
  for $j := 1$ **to** i **do**
    $C_j^l := \mathcal{A}^{\text{lsdc}}\ (\mathcal{V}_j, R_i^l)$ // (Def. 20)
  end for

  // Compute J_d^l with current $\hat{\varrho}^l$
  for $j := 1$ **to** i **do**
    $J_d^l(j\Delta t) := \min\{D^u(k) + \hat{\varrho}^l k\Delta t | k \in [1, j-1]\} - D^u(\infty)$ // (Thm. 8)
  end for
end for
$C_i := C_i^L$

Proof:(1) The first clock C^1 is computed by an LS-CSA, as $\mathcal{A}^{\text{lsdc}}(\mathcal{V}_j, \hat{\rho})$ is equivalent to $\mathcal{A}^{\text{ls}}(\mathcal{V}_j)$, which is an LS-CSA (by Lem. 1). (2) Every J_d^l computed by Alg. 4 is an upper bound on the peak jitter if clock C^l is computed by an LS-CSA and $-\hat{\varrho}^l$ is a lower bound on ρ (by Thm. 8). In particular, this is the case for J_d^1, since (1) and $-\hat{\varrho}^1 = -2\hat{\rho}$. (3) R_i^{l+1} is an upper bound on ρ and $-\hat{\varrho}^l$ is a lower bound on ${\varrho_i}^l$ if peak jitter of clock C^l is bounded by J_d^l (by Thm. 11). This is the case because of (2). (4) The clock C^{l+1} is computed by an LS-CSA

if R_i^{l+1} is an upper bound on ρ, which is the case since (3). Arguments (2)–(4) prove by induction that every clock C^l is computed by an LS-CSA, thus C is computed by an LS-CSA, and algorithm $\mathcal{A}^{\mathrm{lsdc,iter}}$ is an LS-CSA. □

Ex. 11: *The bounds on the peak jitter J_d^l and logical clock drift $\hat{\varrho}^l$ computed by algorithm $\mathcal{A}^{\mathrm{lsdc,iter}}$ can be used to compute approximate bounds achieved by this algorithm. We use*

$$\hat{\varrho}^l = \begin{cases} 2\hat{\rho} & \text{if } l = 1 \\ \min\{\frac{2J_d^{l-1}(j\Delta t)}{(i-j)\Delta t} \mid j \in [1, i-1]\} & \text{otherwise} \end{cases} \tag{3.17}$$

$$J_d^l(i\Delta t) = \min\{D^u(j) + \hat{\varrho}^l j\Delta t \mid j \in [1, i-1]\} \tag{3.18}$$

to compute the upper bound on the peak jitter $J_d^l(i\Delta t)$ in the l-th iteration and after setup time $i\Delta t$. Note that in Eq. 3.17, we have used $(i-j)\Delta t$ where $\mathcal{A}^{\mathrm{lsdc,iter}}$ computed $C^{l-1}(h_i) - C^{l-1}(h_j) + J_d^{l-1}(j\Delta t)$. Both expressions are upper bounds on the length of the real-time interval $[t_j, t_i]$. The form $(i-j)\Delta t$ provides a more pessimistic $\hat{\varrho}^l$, but does not require that the clock C^{l-1} is actually computed based on a concrete view.

The same delay-interval curve as in Ex. 9 is used. Fig. 20, left side shows the bounds J_d^l for $l \in [2, 5]$. On the right side, the bounds computed under the assumption of variable drift are shown. The figure illustrates that under variable drift, drift compensation cannot be done with an arbitrarily small error, even after a long setup time. We can now also verify the rule of thumb presented in Ex. 10. As drift variation was $\vartheta = 10^{-7}\mathrm{s}^{-1}$, the predicted setup time is $140\mathrm{s}$, corresponding quite well to the curve shown in Fig. 20.

Discussion

Algorithm $\mathcal{A}^{\mathrm{lsdc,iter}}$ is not really practical because it is prohibitively expensive in terms of memory and computation overhead and also because it requires that an upper-delay interval curve D^u is known to the algorithm. The purpose of presenting such an impractical algorithm is (i) to illustrate the use of Theorems 8 and 11 for reducing peak jitter and accuracy in comparison with the simple Local-Selection algorithm $\mathcal{A}^{\mathrm{ls}}$ (Fig. 20, left side and Fig. 19), (ii) to show that compensation of constant drift is possible and arbitrarily small peak jitter can be guaranteed after a sufficiently long setup time, and (iii) to establish a yardstick for assessing the performance of much simpler heuristic algorithms presented in the next section.

A similar algorithm can be designed for the assumption of variable drift. Instead of Thm. 11, the computation of the drift estimate employs Thm. 12. The peak jitter achieved by such an algorithm is shown in Fig. 20, right side.

Summary

In this section, we have shown that in principle, it is possible to construct LS-CSAs that compensate clock drift. Algorithm $\mathcal{A}^{\mathrm{lsdc,iter}}$ has been presented,

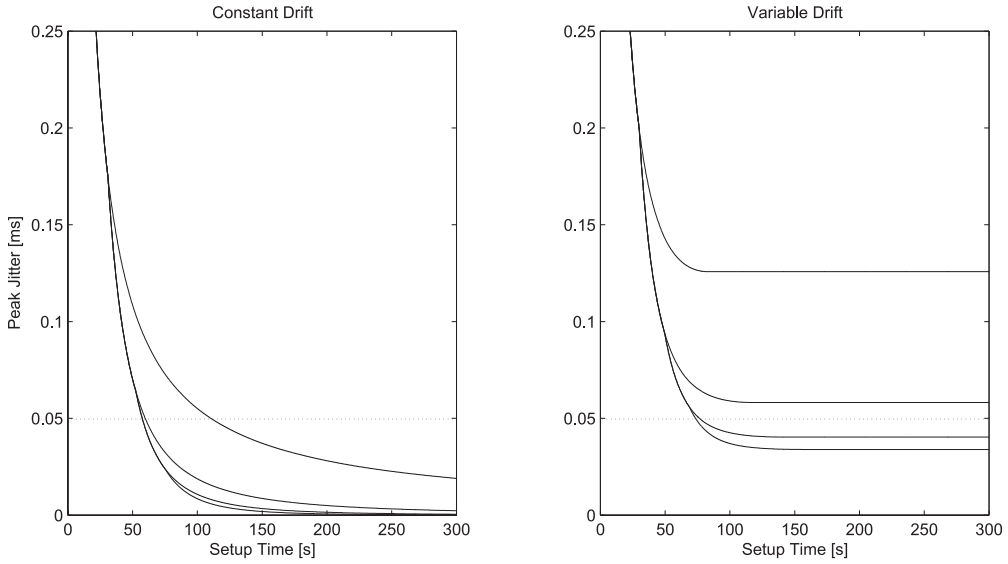


Fig. 20: On the left, the upper bound on the peak jitter achieved by $\mathcal{A}^{\mathrm{lsdc,iter}}$ is shown for $2-5$ iterations. The bound for 1 iteration (corresponding to $\mathcal{A}^{\mathrm{ls}}$) is shown in Fig. 19. While the first iteration reduces the bound on peak jitter to $770\mu\mathrm{s}$ after $2\mathrm{s}$, the drift compensation mechanism further reduces this bound to less than $50\mu\mathrm{s}$ after $110\mathrm{s}$ in the second iteration. In the third iteration, this is achieved already after $60\mathrm{s}$. Note that in all iterations, the peak jitter goes to zero after a sufficiently long setup time. On the right, the upper bound on the peak jitter achieved by a similar algorithm for variable drift is shown. Here, four iterations are needed to get a bound on peak jitter of $50\mu\mathrm{s}$ achieved after $70\mathrm{s}$.

which estimates clock drift with arbitrarily small error for a sufficiently large setup time (by Thm. 13). It has been shown that this is not possible if the clock drift is variable, and a simple rule of thumb has been presented to compute the achievable error of drift compensation and the corresponding setup time (Ex. 10). With a numerical example (Ex. 11, Fig. 20), we have shown that drift compensation can dramatically reduce the deterministic upper bound J_d on the peak jitter compared to LS-CSAs without drift compensation (Fig. 19), though only after a relatively long setup time.

3.5 Local-Selection Heuristics

This section presents *practical* variants of the Local-Selection principle. While also algorithm $\mathcal{A}^{\mathrm{lsdc,iter}}$ presented in the previous section is *feasible*, it is not really practical:

- **Parameters:** A practical CSA should not require knowledge about message delays (e.g. $d_{\min}$ and $d_{\max}$ or D^u). While it is mandatory to have this knowledge

to compute upper bounds on performance metrics in an offline estimation, the CSA itself should not use such knowledge in its computations.

- **Memory:** A practical CSA should only access a small subset of the view $\mathcal{V}_i$, as the size of the total view quickly becomes large. E.g. for $\mathcal{V}_i = \{(s_j, h_j)|j \leq i\}$ with $\Delta t = 10\text{ms}$ and a time-stamp size of 8 bytes, the size of the view after 10 minutes is approximately 940 kilobytes. For embedded systems (see the application scenario described in Sect. 3.6), this is not tolerable.

- **Computation Overhead:** In embedded systems, not only memory is restricted but also the available computation resources. Clock synchronization is hardly ever the main purpose of a platform, and thus its computation overhead should be as small as possible. This is not the case with algorithm $\mathcal{A}^{\text{lsdc,iter}}$, which e.g. executes many minimum-operations over large arrays.

It is certainly possible to find a more efficient version of $\mathcal{A}^{\text{lsdc,iter}}$ that stores intermediate results between consecutive executions (for computing C_i and C_{i+1}), thus the second and third argument why $\mathcal{A}^{\text{lsdc,iter}}$ is not practical may not be very strong. But then, the first argument already suffices.

In the following, we first show that it is not possible to construct an LS-CSA other than $\mathcal{A}^{\text{ls}}$ if the algorithm has no knowledge about message delays. We thus introduce the class of heuristic Local-Selection algorithms (HLS-CSAs), which do not comply with the slow-logical-clock property (Prop. 2). Instead they fulfill the eventually-slow-logical-clock property and thus are live (but not optimally selective).

We then present two concrete examples of HLS-CSAs. We compare the achieved peak jitter of these algorithms with the upper bound J_d of algorithm $\mathcal{A}^{\text{lsdc,iter}}$. While we have no guarantee that the performance of the HLS-CSAs is not much worse on some peculiar delay sequences, they outperform the correct $\mathcal{A}^{\text{lsdc,iter}}$ in typical cases.

3.5.1 Lower Bound on Drift Compensation

Let a view $\mathcal{V}_i = \{(s_j, h_j)|j \leq i\}$ but no additional information about message delays. Clearly, algorithm $\mathcal{A}^{\text{lsdc,iter}}$ cannot be used because it requires the upper delay-interval curve D^u. But maybe there exists some other method to estimate clock drift, such that the resulting synchronized clock C has the properties of a synchronized clock computed by an LS-CSA?

Thm. 14: (Drift and Delay Compensation is impossible) *(i) No deterministic CSA can compute from the view $\mathcal{V}_i = \{(s_j, h_j)|j \leq i\}$ an upper bound R on the drift ρ that is smaller than $\hat{\rho}$. (ii) No deterministic CSA can compute from this view $\mathcal{V}_i$ a lower bound D on the message delays d_i that is larger than 0.*

Proof: The proof of the theorem is by contradiction. Assume a CSA computes $R < \hat{\rho}$ or $D > 0$. We can construct a delay sequence assuming $\rho = \hat{\rho}$ and $d_{\min} = 0$ that provides the CSA with exactly the same view as the original

delay sequence assuming a constant drift of $\rho = R$ and $d_{\min} = D$. As the CSA is deterministic, it computes the same R and D, but these are not valid.

For simplicity, we assume a constant drift ρ. Clearly, if drift compensation is not possible in the case of constant drift, then it is also not possible in the case of variable drift. Let $R = \rho$ be the upper bound on this drift computed by the CSA. The drift ρ is given by $\rho = \frac{h_j - h_{j-1}}{t_j - t_{j-1}} - 1$ for any $j \in [1, i]$. Remember that the reference time at reception of a time-stamp message t_j is the send time of this message plus its delay ($t_j = s_j + d_j$). We can compute modified delays d'_j and d'_{j-1}, such that the drift becomes $\rho' = \hat{\rho}$ and the view $\mathcal{V}_j$ remains unchanged: From $\rho' = \frac{h_j - h_{j-1}}{(s_j + d'_j) - (s_{j-1} + d'_{j-1})} - 1$ we get $d'_j = \frac{h_j - h_{j-1}}{\rho' + 1} + s_{j-1} + d'_{j-1} - s_j$, and from $\rho = \frac{h_j - h_{j-1}}{(s_j + d_j) - (s_{j-1} + d_{j-1})} - 1$ we get $d_j = \frac{h_j - h_{j-1}}{\rho + 1} + s_{j-1} + d_{j-1} - s_j$. Combining the two equations results in $d'_j - d_j = d'_{j-1} - d_{j-1} - (h_j - h_{j-1}) \frac{\rho' - \rho}{(1+\rho')(1+\rho)}$. With $\Delta d_j = d'_j - d_j$, $\Delta d_{j-1} = d'_{j-1} - d_{j-1}$, and using the approximation $\frac{1}{(1+\rho')(1+\rho)} \approx 1$ we arrive at $\Delta d_{j-1} = \Delta d_j + (h_j - h_{j-1})(\rho' - \rho)$. This equation implies that in order to get the delay sequence that produces the view $\mathcal{V}_i$ but with $\rho' = \hat{\rho}$, the j-th delay has to be shortened less than the $j-1$-th delay (since $(h_j - h_{j-1})(\hat{\rho} - \rho)$ is positive). Thus we construct an alternative trace, in which we have $t'_i = s'_i = s_i$ (thus with $d'_i = d_{\min} = 0$) and $t'_j = s_j + d'_j$, where $d'_j = d_j + \Delta d_j$ and $\Delta d_j = \Delta d_{j+1} + (h_{j+1} - h_j)(\rho' - \rho)$ for all $j \in [1, i-1]$. The view $\mathcal{V}_i$ remains unchanged, but $\rho' = \hat{\rho}$ and $d_{\min} = 0$. □

In a sense, this result is frustrating, because it tells us that we cannot hope to construct a better LS-CSA than algorithm $\mathcal{A}^{\text{ls}}$ if we have no knowledge about message delays. On the other hand, this result is also liberating: if not even very complicated procedures (see Alg. 4) can guarantee that some drift estimation R is an upper bound on ρ, then we should use simple procedures instead and just see empirically how they perform.

Def. 21: (Heuristic LS-CSAs) *A selective estimate-based CSA $\mathcal{A}$ is a Heuristic Local-Selection CSA (HLS-CSA) if Properties 1 and 3 hold, and the drift ϱ_i of every logical clock* eventually *becomes negative, i.e.*

$$\exists t_i^* \geq t_i : \quad -\hat{\varrho} < \varrho_i(t) < 0 \quad \forall i, \forall t \geq t_i^*$$

Lem. 15: (Safety and Liveness) *Every HLS-CSA is safe and live.*

Proof: The proof of safety is the same as in the case of $\mathcal{A}^{\text{ls}}$ (Lem. 4). The proof of liveness is the same as for LS-CSAs (Lem. 5), since the error of every logical clock C_i eventually is smaller than e^*. But for an HLS-CSA, it possibly takes more time than for an LS-CSA. □

An HLS-CSA is not optimally selective. As logical clocks initially may have a positive drift ϱ_i, the synchronized clock C can have a positive error. If for some j, $e_i^* < 0$ and $e_i^- > 0$, then the candidate clock C_i^* is not selected, even though $|e_i^*|$ may be smaller than $|e_i^-|$.

Def. 22: (Leakage Local Selection) *The Leakage Local-Selection algorithm $\mathcal{A}^{\mathrm{leak}}$ is a selective estimate-based CSA that computes the candidate logical clock C_i^* and decision π_i from the view $\mathcal{V}_i = \{(s_j, h_j) | j \leq i\}$ according to the following rule:*

$$C_i^*(h) = s_i + \frac{h - h_i}{1 + R_i(h)}, \quad \forall h \geq h_i \tag{3.19}$$

$$\pi_i = (c_i^* > c_i^-), \quad \forall i > 1 \tag{3.20}$$

The function R_i is computed as

$$R_i(h) = r_i + \lambda(h - h_i) \tag{3.21}$$

where the constants r_i and λ are parameters either a-priori specified or computed from the view $\mathcal{V}_i$.

Lem. 16: ($\mathcal{A}^{\mathrm{leak}}$ is an HLS-CSA) *Algorithm $\mathcal{A}^{\mathrm{leak}}$ is an HLS-CSA if $\lambda > \frac{\hat{\vartheta}}{2(1-\hat{\rho})}$.*

Proof: We have to show that $\varrho_i(t) < 0$ for some $t \geq t_i$. We know that $(1-\hat{\rho})(t-t_i) < (h-h_i)$. We also know that $(h-h_i) < (1+\rho(t_i))(t-t_i) + \frac{1}{2}\hat{\vartheta}(t-t_i)^2$. Thus, $C_i(h) < s_i + \frac{(1+\rho(t_i))(t-t_i)+\frac{1}{2}\hat{\vartheta}(t-t_i)^2}{1+r_i+\lambda(1-\hat{\rho})(t-t_i)} = s_i + (t-t_i)\frac{(1+\rho(t_i))+\frac{1}{2}\hat{\vartheta}(t-t_i)}{1+r_i+\lambda(1-\hat{\rho})(t-t_i)}$. With $\lambda > \frac{\hat{\vartheta}}{2(1-\hat{\rho})}$, we get $C_i(h) < s_i + (t-t_i)\frac{(1+\rho(t_i))+\frac{1}{2}\hat{\vartheta}(t-t_i)}{1+r_i+\frac{1}{2}\hat{\vartheta}(t-t_i)}$. Let $(t-t_i) \to \infty$, then $C_i(h) < s_i + (t-t_i)$. On the other hand, we have $C_i(h) = s_i + \int_{t_i}^{t}(1+\varrho_i(t))\mathrm{d}t$. Thus, $\int_{t_i}^{t}(1+\varrho_i(t))\mathrm{d}t < (t-t_i)$ and consequently $\varrho_i(t) < 0$. □

Lemma 16 implies that algorithm $\mathcal{A}^{\mathrm{leak}}$ is safe and live, no matter whether the initial drift estimation r_i is an upper bound on $\rho(t_i)$ or not. If it is not, then the logical clock C_i progresses faster than real time for some time, but eventually its drift ϱ_i becomes negative again (an example is shown in Fig. 21, right column).

In the following, we present two different strategies for constructing HLS-CSAs.

3.5.2 Strategy 1: Approximate Local Selection

In the previous section, we have shown, that an LS-CSA with drift compensation can be constructed if a bound on the peak jitter J is known. We had used information about message delays to derive such a bound J. If nothing is known about message delays, such a procedure is not possible. Instead, a bound on peak jitter can be estimated. This is done in the Approximate Local-Selection algorithm, shown in Alg. 5.

Rationale

In the first ι executions, algorithm $\mathcal{A}^{\mathrm{leak,app}}$ is essentially equivalent to $\mathcal{A}^{\mathrm{ls}}$; the drift of the local clock is not compensated.

In the queues, information about logical clocks that are different from their predecessors is stored: In queue Q^j, the difference between the initial value s_i

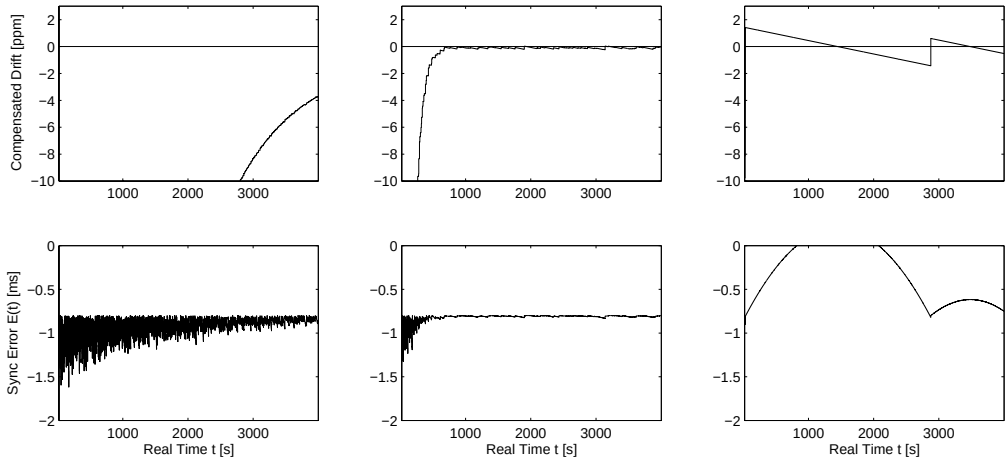


Fig. 21: *Parameters of the Agnostic Local-Selection Algorithm:* The top row displays the drift ϱ of the synchronized clock computed by the Agnostic LS algorithm with three compensation factors α. The bottom row shows the corresponding synchronization errors. On the left, α is chosen too small - drift compensation is very slow. On the right, α is far too large; the drift is over-compensated, resulting in periods where no candidate clocks are selected. In the middle, the choice of α is accurate; the synchronized clock has less than $0.3\mathrm{ppm}$ drift after $600\mathrm{s}$ setup time. The synchronization error shown in this figure can be compared to the synchronization error achieved without drift compensation, shown in Fig. 18. There the synchronization is approximately contained in the interval $[-2.5\mathrm{ms}, -800\mu\mathrm{s}]$, here (middle column), it lies in $[-840\mu\mathrm{s}, -800\mu\mathrm{s}]$ after approximately $300\mathrm{s}$ setup time.

and the current time of the previous logical clock c_i^- is stored, queue Q^h stores the local time h_i, and queue Q^c memorizes the synchronized time s_i at the start of this logical clock.

If these queues are full, then the peak jitter is estimated as the largest difference in queue Q^j. With this estimate and Eq. 3.12, the drift is estimated over the interval defined by the oldest event in the queues and the current event (local-time interval $[Q^h.tail, h_i]$, synchronized-time interval $[Q^c.tail, s_i]$).

Finally, the Leakage Local-Selection algorithm is executed with the computed drift estimation r_i and the a-priori specified leakage factor λ.

Parameters

Choosing the parameters for algorithm $\mathcal{A}^{\mathrm{leak,app}}$ incurs various trade-offs. In the following, the general lines that have to be considered when choosing these parameters are discussed. Figure 22 shows how evolutionary optimization (see Sect. 2.7) determines these parameters. This optimization evaluates the performance on traces recorded in an 802.11b WLAN in ad-hoc mode under heavy network load. A detailed description follows in Sect. 3.5.5. Note that we do not count the maximal drift of the local clock $\hat{\rho}$ and its maximal variation $\hat{\vartheta}$ as parameters of the algorithm.

Algorithm 5 Approximate Local Selection $\mathcal{A}^{\text{leak,app}}$

Parameters: Initial phase ι, leak factor λ, queue size q
Input: View $\mathcal{V}_i = \{(s_j, h_j) | j \in [1, i]\}$
Output: Logical clock C_i
Variables: FIFO queues Q^j, Q^h, Q^c of size q

```
if i > ι then
    r_i := R_{i-1}(h_i)
    if s_i − c_i^− > 0 then
        insert (s_i − c_i^−) at head of Q^j
        insert s_i at head of Q^c
        insert h_i at head of Q^h
        if Q^j is full then
            r_i := (h_i − Q^h.tail) / (s_i − Q^c.tail − max(Q^j)) + (1/2) ϑ̂ (s_i − Q^c.tail + max(Q^j)) − 1
        end if
    end if
else
    r_i := ρ̂
end if
C_i := A^leak(V_i, r_i, λ)   //(Def. 22)
```

- **Initial phase ι.** The estimation of the peak jitter is not accurate for the first messages. Therefore, these initial messages should not be used for drift compensation. But delayed drift compensation leads to an increased setup time. The evolutionary optimization (Fig. 22) resulted in $\iota = 12$.

- **Queue size q.** Clearly, the memory requirements of the algorithm directly increase with q. On the other hand, the estimation of the peak jitter improves if the queues contain more elements. Also the time interval over which drift is estimated becomes larger with increasing q and thus reduces the negative influence of an inaccurate estimation of the peak jitter. On the other hand, in the case of strongly variable drift, this interval may become too large. The evolutionary optimization (Fig. 22) resulted in a surprisingly small $q = 6$.

- **Leakage factor λ.** This parameter determines how fast the logical clocks decelerate (compare Eq. 3.21). If λ is chosen much larger than $\frac{\hat{\vartheta}}{2(1-\hat{\rho})}$ (compare Lem. 16), then a wrong estimation r_i of the local clock drift is quickly corrected. On the other hand, a good estimation cannot be maintained for a long time. The evolutionary optimization (Fig. 22) resulted in $\lambda = 7 \cdot 10^{-8}\text{s}^{-1} = 260\text{ppm/h}$.

3.5.3 Strategy 2: Agnostic Local Selection

The first strategy for constructing a heuristic Local-Selection algorithm was to mimic algorithm $\mathcal{A}^{\text{lsdc}}$ and estimate the missing piece of information at run time. We now present a second approach, which completely ignores the theory developed in previous sections, in order to simplify the procedures. The algorithm

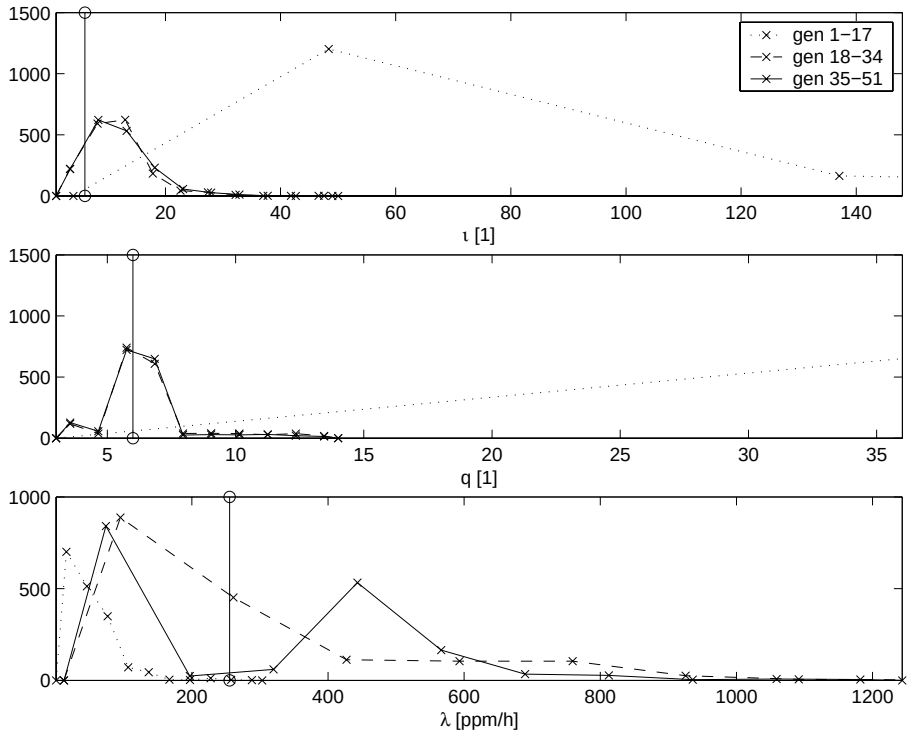


Fig. 22: Distribution of the Approximate Local-Selection algorithm's parameters during the evolutionary optimization process. Every cross represents a set of solutions for which the parameter has the value shown on the x-axis. The number of solutions in this set is shown on the y-axis. The dotted line represents the first 17 generations containing 100 parameter sets each. The dashed line represents generations 18 to 34 and the solid line generations 35 to 50. The vertical line shows the parameters of the best individual. The initial phase ι and the queue size q become smaller during the optimization process, the leakage factor λ increases.

presented in Alg. 6 is thus dubbed Agnostic Local Selection.

Algorithm 6 Agnostic Leakage Local Selection $\mathcal{A}^{\text{leak,agn}}$

Parameters: Initial phase ι, leak factor λ, drift compensation factor α
Input: View $\mathcal{V}_i = \{(s_j, h_j) | j \in [1, i]\}$
Output: Logical clock C_i

if $i > \iota$ **then**
  $r_i := R_{i-1}(h_i)$
  if $s_i - c_i^- > 0$ **then**
    $r_i := R_{i-1}(h_i) - \alpha(s_i - c_i^-)$
  end if
else
  $r_i := \hat{\rho}$
end if
$C_i := \mathcal{A}^{\text{leak}}(\mathcal{V}_i, r_i, \lambda)$ //(Def. 22)

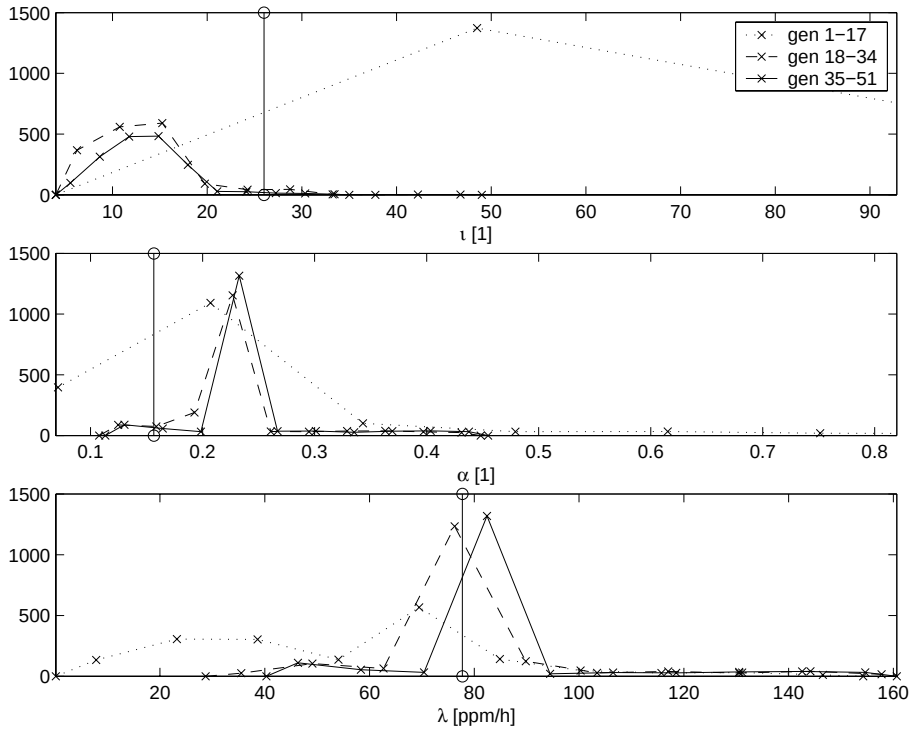


Fig. 23: Distribution of Agnostic Local-Selection parameters during the evolutionary optimization process. Every cross represents a set of solutions for which the parameter has the value shown on the x-axis. The number of solutions in this set is shown on the y-axis. The initial phase ι is slightly longer, the leakage factor is smaller than in the case of the Approximate LS algorithm (see Fig. 23).

Rationale

The initial behavior of Algorithm $\mathcal{A}^{\mathrm{leak,agn}}$ is equivalent to $\mathcal{A}^{\mathrm{leak,app}}$. Whenever a candidate clock is selected ($s_i > c_i^-$), it is quite likely that the drift of the latest logical clock is negative. Thus, the algorithm sets the new estimation r_i to a smaller value. The adjustment is proportional to the difference between the current reading of the previous logical clock C_{i-1} and the candidate clock $c_i^* = s_i$.

Parameters

Two of the parameters, the initial phase ι and the leakage factor λ are the same as for algorithm $\mathcal{A}^{\mathrm{leak,app}}$. The results of evolutionary parameter optimization on traces recorded in an 802.11b WLAN in ad-hoc mode under heavy network load are shown in Fig. 23. The initial phase ι is 26, slightly more than for $\mathcal{A}^{\mathrm{leak,app}}$. The leakage factor resulting from optimization is $\lambda = 2 \cdot 10^{-8}\mathrm{s}^{-1} = 78\mathrm{ppm/h}$, thus slightly smaller than for $\mathcal{A}^{\mathrm{leak,app}}$.

In addition, the Agnostic LS algorithm has a drift compensation factor α:

- **Drift compensation factor α.** Figure 21 illustrates the effect of the parameter α. A large α causes aggressive drift compensation, i.e. it happens quite often

that the resulting r_i is too small and thus the drift of the logical and the synchronized clocks is positive. A small α on the other hand leads to inefficient drift compensation. After evolutionary optimization (Fig. 23), the drift compensation factor is $\alpha = 0.16\mathrm{s}^{-1}$

3.5.4 Adaptive Parameters

Algorithms $\mathcal{A}^{\mathrm{leak,app}}$ and $\mathcal{A}^{\mathrm{leak,agn}}$ present simple ideas for the estimation of clock drift. More complicated mechanisms may improve the performance, but also make the algorithm more difficult to understand and also to implement. In the following, two reasonably simple modifications are presented.

The purpose of decelerating the logical clocks is twofold: First, it should be avoided that a change of the local clock rate leads to fast logical clocks. Second, it is supposed to correct over-compensation of the clock drift (see Fig. 21). As the latter occurs frequently at the start of a time-stamp-message sequence, the parameter λ should initially be considerably larger than $\frac{\hat{\vartheta}}{2(1-\hat{\rho})}$, but can decrease over time. This parameter thus should not be a constant. Instead,

$$\lambda := (1 - \lambda_\mu)\lambda + \lambda_\mu \lambda_{\min} \tag{3.22}$$

can be applied whenever a logical clock is different from its predecessor, i.e. if $(s_i > c_i^-)$. This introduces two new parameters: $\lambda_\mu \in [0, 1]$, which describes how quickly the leakage factor is reduced, and $\lambda_{\min}$, which specifies a lower threshold for the leakage factor. Evolutionary optimization of these parameters for algorithm $\mathcal{A}^{\mathrm{leak,app}}$ resulted in $\lambda = 8 \cdot 10^{-7}\mathrm{s}^{-1} = 3000\mathrm{ppm/h}$, $\lambda_\mu = 0.3$ and $\lambda_{\min} = 2 \cdot 10^{-12}\mathrm{s}^{-1} = 0.2\mathrm{ppm/day}$ [2].

In the case of algorithm $\mathcal{A}^{\mathrm{leak,agn}}$, the same argument applies to the drift compensation factor α. The factor may be adapted by

$$\alpha := (1 - \alpha_\mu)\alpha + \alpha_\mu \alpha_{\min} \tag{3.23}$$

Here the values $\alpha = 0.5\mathrm{s}^{-1}$, $\alpha_\mu = 0.2$ and $\alpha_{\min} = 0.003\mathrm{s}^{-1}$ have been found by the optimization process. The complete adaptive variants of algorithms $\mathcal{A}^{\mathrm{leak,app}}$ and $\mathcal{A}^{\mathrm{leak,agn}}$ are stated in the appendix.

3.5.5 Experimental Study: Comparison of Heuristic Algorithms

We have presented two heuristic LS-CSAs and their adaptive variants. In this section, we evaluate and compare the performance and cost of these heuristics.

Experimental Setup

Four algorithms are evaluated, $\mathcal{A}^{\mathrm{leak,app}}$ (Sect. 3.5.2), $\mathcal{A}^{\mathrm{leak,agn}}$ (Sect. 3.5.3) and their adaptive variants (sketched in Sect. 3.5.4).

In a first step, the parameters of these algorithms are optimized, using the procedure described in Sect. 2.7. Populations of 100 individuals are optimized

[2] Compare the values for λ and $\lambda_{\min}$ with $\hat{\vartheta}$ of Ex. 10. The values found by the evolutionary optimization nicely mirror the measured drift variations.

over 50 generations, thus 5000 parameter sets are evaluated. The evaluation is based on recorded traces (see Sect. 2.6.1) from an 802.11b WLAN in ad-hoc mode. Three traces with a message interval of $\Delta t \approx 20\text{ms}$, containing 50000 messages each are used. The goals of the optimization are (i) to minimize the peak jitter after 10 seconds and (ii) to minimize the setup time when a peak jitter of $100\mu\text{s}$ is achieved. The parameter set with the best peak jitter after 10 seconds was selected as the best parameter set. The results of this optimization step for $\mathcal{A}^{\text{leak,app}}$ and $\mathcal{A}^{\text{leak,agn}}$ are shown in Figs. 22 and 23. The parameters of the adaptive variants are described in Sect. 3.5.4.

The optimized algorithms are evaluated on ten traces with the same parameters as those used in the optimization step. Figures 24 and 25 display the synchronization error, the peak jitter, and the drift of the synchronized clock achieved by algorithms $\mathcal{A}^{\text{leak,app}}$ and $\mathcal{A}^{\text{leak,agn}}$ on the trace where the peak jitter after 10 seconds is largest. In Tab. 7, the worst peak jitter and setup time achieved by the four algorithms on the ten traces are summarized.

Table 8 shows the cost in terms of computation and memory overhead of these algorithms. The memory cost represents the number of symbols used by the algorithm. This calculation is illustrated using the example of algorithm $\mathcal{A}^{\text{leak,app}}$: The algorithm uses five constants ($\iota, \lambda, q, \hat{\rho}, \hat{\vartheta}$) and five time stamps from the view (h_i, s_i, r_i and $h_{i-1}, s_{i-1}, r_{i-1}$ in the form of R_{i-1} and c_{i-1}^-). Additionally, the variable i and the queues are used, resulting in $3 \cdot q + 1$ symbols. For $q = 6$ determined by the optimization process, we get the total of 30 symbols. The computation overhead is divided into categories: Additions and subtractions (seven for $\mathcal{A}^{\text{leak,app}}$, i.e. one for $s_i - c_i^-$ and six for the computation of r_i), multiplications (two for the computation of r_i and one for R_{i-1}), divisions (one for the computation of r_i and one for c_i^-) and comparisons (three: $i > \iota$, $s_i > c_i^-$ and $Q^j\text{full}$).

Algorithm	*Peak Jitter*		*MTIE*	
	After 10s	$S(J \leq 100\mu\text{s})$	*After* 10s	$S(M \leq 10\mu\text{s})$
$\mathcal{A}^{\text{leak,app}}$	$18\mu\text{s}$	0.1s	$14\mu\text{s}$	23s
$\mathcal{A}^{\text{leak,app,adap}}$	$21\mu\text{s}$	0.1s	$2\mu\text{s}$	4s
$\mathcal{A}^{\text{leak,agn}}$	$49\mu\text{s}$	0.2s	$25\mu\text{s}$	–
$\mathcal{A}^{\text{leak,agn,adap}}$	$38\mu\text{s}$	0.2s	$17\mu\text{s}$	27s
$\mathcal{A}^{\text{lsdc,iter}}$	$70\mu\text{s}$	8s	$54\mu\text{s}$	67s

Tab. 7: *Performance of heuristic LS-CSAs.* The table shows the worst performance values achieved by the algorithms on ten traces. The peak jitter J and the MTIE M with $\tau = 10\text{s}$ are measured after 10s. Also the setup times after which the algorithms achieve a peak jitter of less than $100\mu\text{s}$ and an MTIE of less than $10\mu\text{s}$ are evaluated. The Agnostic LS does not achieve $M \leq 10\mu\text{s}$. The Iterative Drift-Compensation algorithm is shown as a yardstick for the heuristic algorithms.

Discussion

Remember that algorithm $\mathcal{A}^{\text{leak,app}}$ makes use of the theory that was developed

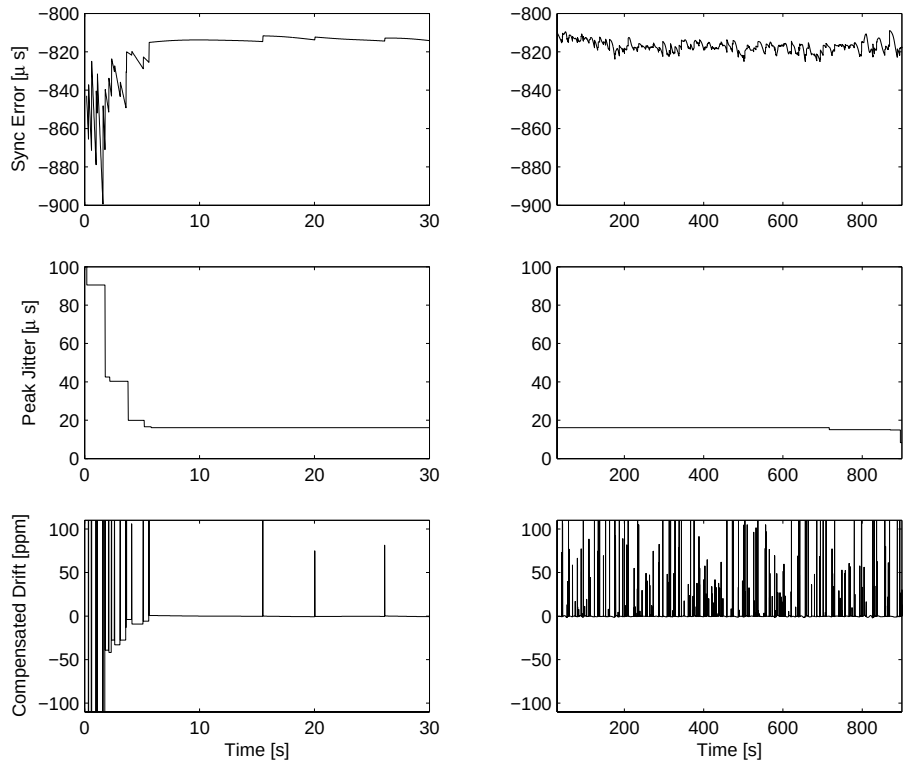


Fig. 24: Performance of the optimized Approximate Local-Selection algorithm in an 802.11b WLAN in ad-hoc mode with heavy network load. The left column displays the initial 30 seconds, the right column shows the following 15 minutes. The top row displays the synchronization error, in the middle the peak jitter is shown and the bottom row shows the drift of the synchronized clock. The spikes in the drift curves are due to the non-continuous nature of the synchronized clock.

Algorithm	*Operations*				*Memory*
	n^+	$n^<$	n^*	$n^/$	
$\mathcal{A}^{\text{leak,app}}$	7	3	3	2	30
$\mathcal{A}^{\text{leak,app,adap}}$	9	3	5	2	32
$\mathcal{A}^{\text{leak,agn}}$	2	2	1	–	11
$\mathcal{A}^{\text{leak,agn,adap}}$	6	2	5	–	15

Tab. 8: *Computation and memory overhead of heuristic LS-CSAs.* The computation overhead (*Operations*) represents the number of operations executed upon reception of a time-stamp message. The operations are categorized into additions/subtractions, comparisons, multiplications, and divisions. The memory overhead is a count of symbols used by the algorithm. A time-stamp typically is ≤ 8 Bytes long, thus the memory overhead of the Approximate LS-CSA is ≤ 240 Bytes.

in Sects. 3.2–3.4, while algorithm $\mathcal{A}^{\text{leak,agn}}$ does not. In Tab. 7, we see that the Approximate Local-Selection algorithm outperforms the Agnostic Local-Selection algorithm in all criteria. This is also the case for the adaptive variants

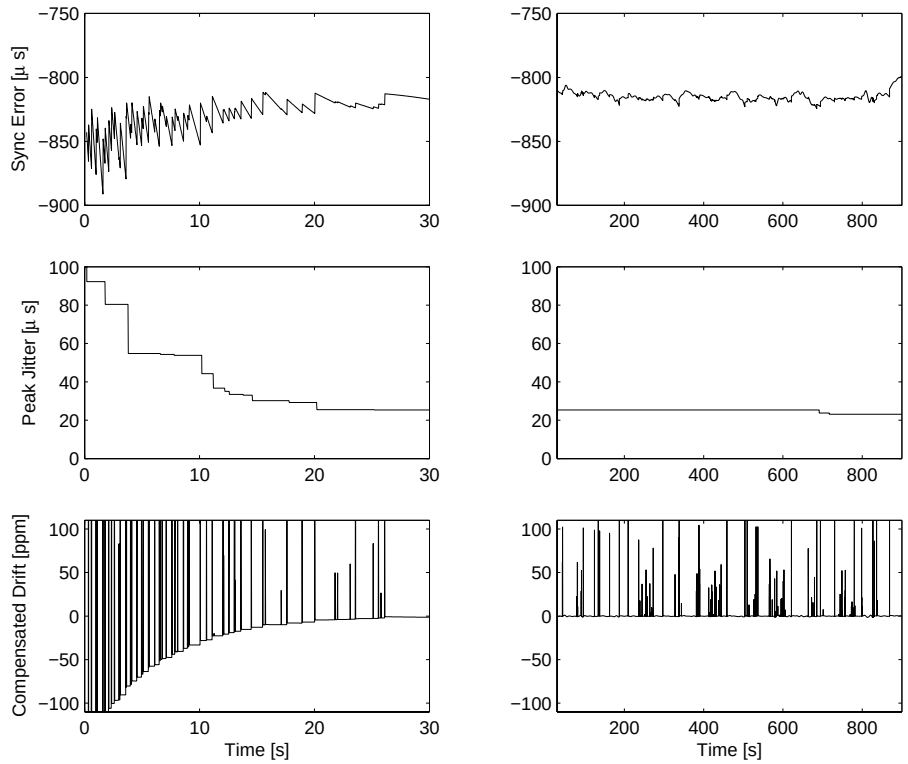


Fig. 25: Performance of the optimized Agnostic Local-Selection algorithm in an 802.11b WLAN in ad-hoc mode with heavy network load. The left column displays the initial 30 seconds, the right column shows the following 15 minutes. The top row displays the synchronization error, in the middle the peak jitter is shown and the bottom row shows the drift of the synchronized clock. The spikes in the drift curves are due to the non-continuous nature of the synchronized clock.

of both algorithms. Thus we conclude that the insights gained in Sects. 3.2–3.4 do help to construct a good heuristic LS algorithm. However, the advantage is not very impressive for the peak jitter. It is more significant for the MTIE, where algorithm $\mathcal{A}^{\mathrm{leak,agn}}$ without adaptive parameters totally fails to achieve the required $10\mu\mathrm{s}$. Also the adaptive variant of $\mathcal{A}^{\mathrm{leak,agn}}$ is more than five times worse ($27\mathrm{s}$ versus $4\mathrm{s}$) than the adaptive variant of algorithm $\mathcal{A}^{\mathrm{leak,app}}$.

We compare the adaptive algorithms with their non-adaptive counterparts. The peak jitter is slightly better for the Agnostic LS, but slightly worse for the Approximate LS. However, the adaptive variants achieve a considerably lower MTIE.

The cost in terms of memory and computation of all algorithms is quite low, as shown in Tab. 8. The adaptive variant of algorithm $\mathcal{A}^{\mathrm{leak,app}}$ is the most expensive, requiring memory for 32 symbols and a total of 19 operations. Algorithm $\mathcal{A}^{\mathrm{leak,agn}}$ requires approximately a third of these resources and does not require divisions. The low complexity of the algorithms permits driver-level or even hardware implementations.

Finally, we compare the performance of the heuristic algorithms with the

exact algorithm $\mathcal{A}^{\mathrm{lsdc,iter}}$. Table 7 shows that in all metrics, the exact algorithm performs much worse than the heuristics. Though this looks surprising at first glance, it can be understood when recalling that the drift compensation of $\mathcal{A}^{\mathrm{lsdc,iter}}$ is guaranteed to be correct, thus necessarily is more prudent (and thus slow) than the drift compensation of the heuristic algorithms. The cost of $\mathcal{A}^{\mathrm{lsdc,iter}}$ is not indicated. The reason is that the cost of the implementation shown in Alg. 4 is enormous[3], but obviously could be reduced considerably.

The conclusion of this experimental section is that the adaptive Approximate LS algorithm performs better at a slightly higher price than the Agnostic LS algorithm. The latter may still be an interesting choice for a hardware implementation, where memory and computation constraints are extremely strong. However, we will use the adaptive Approximate LS algorithm in the experiments presented in the next section.

3.6 Case Study: Wireless Loudspeakers

In this section, the performance of various CSAs for wireless loudspeakers is evaluated and compared. We first present this application and its requirements on clock synchronization. Then the performance is evaluated individually for various network setups and load situations. Finally, we compare the robustness of the evaluated CSAs in changing network setups and variable network load situations.

3.6.1 Application Scenario

In recent years, many new consumer-electronics (CE) technologies and devices have entered our homes. As these devices start to integrate networking capabilities in order to provide new and improved services, wire-bound connections more and more become a burden. Wireless technologies like the IEEE 802.11 standards seem to be a promising alternative, e.g. in home cinema systems that connect a central source with up to eight loudspeakers distributed all over the living room. The use of 802.11b technology provides the additional benefit of an easy integration of living-room CE equipment and the home computer. This would allow the user, for example, to play MP3 files from the Internet directly on the loudspeakers in the living room. Such a setup is illustrated in Fig. 26.

Unfortunately, loudspeakers pose stringent real-time requirements. Variable message delay on the wireless connection and oscillator drift in the loudspeakers are the origins of a temporal offset between the channels played by different loudspeakers[4]. This offset between correlated audio channels can cause unde-

[3]The simulation of a trace of 10 minutes length takes more than 20 minutes on a Pentium4 using non-optimized Matlab code.

[4]Note that this problem is inexistent for conventional analog connections to the loudspeakers, as the propagation time of the signals is constant.

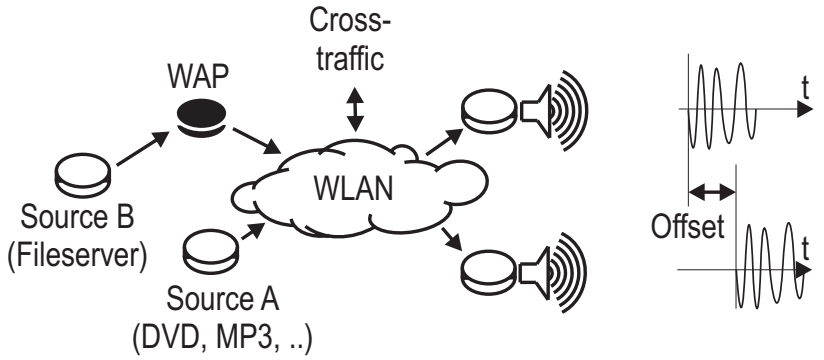


Fig. 26: Case study *Wireless Loudspeakers*: Two or more loudspeakers are connected over a wireless network to an audio source. The source is either directly connected to the wireless network (A) or via a wireless access point (B). A temporal offset between the audio channels causes undesirable psychoacoustic effects.

sirable psychoacoustic effects:

Source Separation. If the temporal offset between the channels is more than ten milliseconds, the loudspeakers are perceived as separate sources.

Timbre Modification. A channel offset of more than one millisecond alters the timbre[5] of the sound.

Directional Perception. A channel offset of more than 100μs influences the directional perception of the sound source.

Noise. Fast fluctuations of the channel offset with a magnitude of at least 10μs can cause noise in the perceived sound.

From these psychoacoustic effects, a set of target metrics is derived. Clearly the offset has to be below one millisecond to avoid source separation and timbre modification. The channel offset is however not required to be less than 100μs. This is because the directional perception of a sound source depends strongly on the physical location of the loudspeakers and anyway has to be adjusted manually (by means of the balance control) to the preferences of the user. However, once this adjustment has been made, it should not have to be modified. A channel offset with an average of say 500μs is perfectly acceptable, as long as it does not vary by more than 100μs. Thus the first three effects are avoided if the accuracy is better than one millisecond and the peak jitter is better than 100μs. Noise generation is avoided if the MTIE in an interval of ten seconds is less than 10μs. A final requirement concerns the setup time. Clearly, the user is not willing to

[5]From Wikipedia, the free encyclopedia: "Timbre (character, color): In music, timbre is the quality of a musical note which distinguishes different types of musical instruments. This is why, with a little practice, you can pick out the saxophone from the trumpet in a jazz group or the flute from the violin in an orchestra, even if they are playing notes at the same pitch and amplitude."

switch on his home cinema ten minutes before using it just to let it synchronize. Thus the aforementioned target metrics have to be met quickly. We use a target setup time of ten seconds. In summary, the application's requirements are described by

$$\begin{aligned} \tilde{S} &= 10\text{s} \\ \tilde{A} &= 1\text{ms} \\ \tilde{J} &= 100\mu\text{s} \\ \tilde{M} &= 10\mu\text{s} \end{aligned}$$

To simplify the following discussion we define an application-specific metric that summarizes the four metrics accuracy, peak jitter, MTIE, and setup time in a single scalar value.

Def. 23: **(Performance Penalty)** *Given the target metrics $\tilde{A}$ for the accuracy, $\tilde{J}$ for the peak jitter, $\tilde{M}$ for the MTIE and $\tilde{S}$ for the setup time, the performance penalty P of an algorithm achieving the accuracy A, the peak jitter J, the MTIE M and the setup time S is*

$$P = \begin{cases} \max\{A/\tilde{A}, J/\tilde{J}, M/\tilde{M}\} & \textit{if } S > \tilde{S} \\ S/\tilde{S} & \textit{otherwise} \end{cases} \tag{3.24}$$

The interpretation of the performance penalty is the following: If after the target setup time all other target metrics are met, then the penalty is ≤ 1. In this case, the penalty gives the information of how much faster than required all target metrics are met. For example $P = 0.3$ means that already after three seconds this is the case. If the target metrics are not met by the target setup time, we have a penalty > 1. In this case, the penalty expresses by what factor the targets are missed. Thus $P = 2$ means that after ten seconds, either the accuracy, the peak jitter, or the MTIE is twice as large as required.

3.6.2 Experimental Setup

In the following, we compare the Adaptive Approximate Local-Selection algorithm $\mathcal{A}^{\text{leak,app,adap}}$(Alg. 8) with the Phase-Locked-Loop CSA (Alg. 3), the Linear-Regression CSA (Alg. 2) and the Gradient CSA (Alg. 10) which is presented in the Appendix. We compare the performance of the CSAs in three different operation modes: (i) Network in ad-hoc mode, (ii) network in infrastructure mode, and the sender is a wireless node (source A in Fig. 26), (iii) network in infrastructure mode, and the sender is part of a wired network that is connected to the wireless network via an access point (source B in Fig. 26). In addition, we vary the network load. An overview of some statistical properties of the traces recorded in these different scenarios is shown in Tab. 3 on page 32.

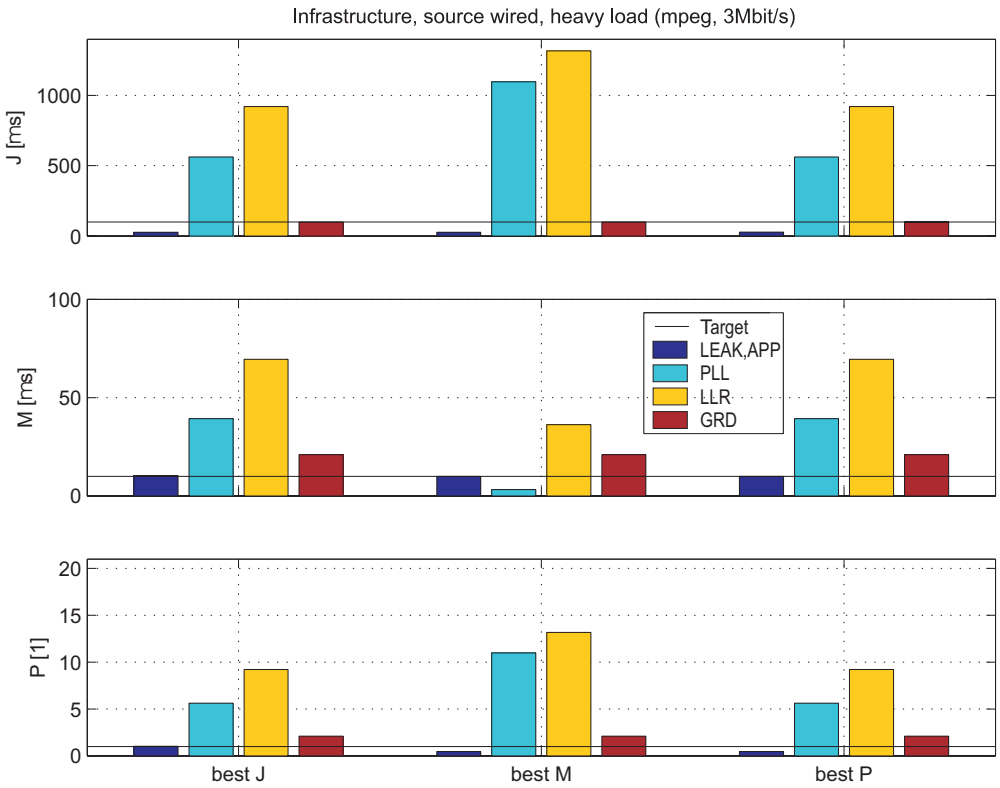


Fig. 27: Achieved peak jitter J, MTIE M, and performance penalty P of three parameterizations per CSA, found by the evolutionary algorithm. The three solutions represent different trade-offs that can be chosen from the set of the non-dominated parameterizations. In most cases, the solution with the best performance penalty is the same as the solution with the best peak jitter.

Which metric is the most difficult to achieve?
In the first experiment, we examine whether different metrics can be conflicting and whether the target peak jitter of 100μs or the target MTIE of 10μs is more difficult to achieve. In Fig. 27, the peak jitter J, the MTIE M, and the performance penalty P of three solutions found by the evolutionary optimization for the infrastructure mode, wired-source scenario with 3MBit/s load are displayed for all algorithms. It can be seen that the peak jitter and the MTIE metrics are conflicting criteria for algorithms $\mathcal{A}^{\text{pll}}$ and $\mathcal{A}^{\text{llr}}$. For the other CSAs, this cannot be shown clearly. Furthermore, the figure shows that for algorithms $\mathcal{A}^{\text{pll}}$ and $\mathcal{A}^{\text{llr}}$, the target peak jitter $\hat{J} = 100\mu$s is more difficult to achieve than the target MTIE $\hat{M} = 10\mu$s, since the solution with the smallest penalty P is identical to the solution with the best peak jitter J. For algorithm $\mathcal{A}^{\text{leak,app,adap}}$, the solution with the smallest penalty P is equivalent to the solution with the smallest MTIE M.

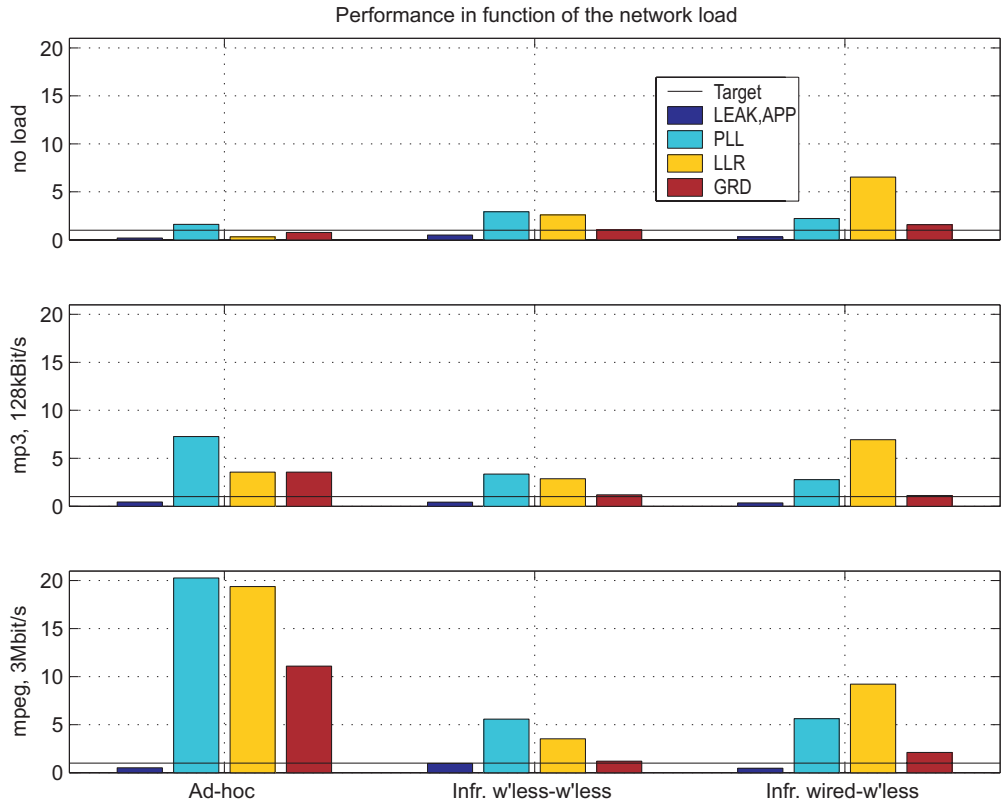


Fig. 28: The performance penalty P of the Adaptive Approximate Local-Selection (leak,app), the Phase-Locked-Loop (pll), the Linear-Regression (llr) and the Gradient (grd) algorithms in the three operation modes and three load situations. The top row shows the results in a network without load, the two rows below show the effect of increasing network load. For every scenario and load type, the parameterization was optimized individually.

Which scenarios and load situations are difficult?

In a second experiment, we were interested in the performance penalty of the CSAs for the various scenarios and load situations. The parameters of the CSAs are optimized specifically for every scenario and every load situation. Figure 28 shows the achieved performance penalties P. It can be seen that for all algorithms and scenarios, the penalty P increases with increasing network load. The negative impact of the network load is more accentuated in ad-hoc scenario than in the infrastructure scenarios. Algorithm $\mathcal{A}^{\text{leak,app,adap}}$ achieves all target metrics in all scenarios and load types (resulting in $P < 1$). Algorithm $\mathcal{A}^{\text{grd}}$ achieves an almost sufficient penalty P in all infrastructure scenarios. Algorithms $\mathcal{A}^{\text{pll}}$ and $\mathcal{A}^{\text{llr}}$ clearly fail to achieve a penalty $P \leq 1$ in all scenarios with network load.

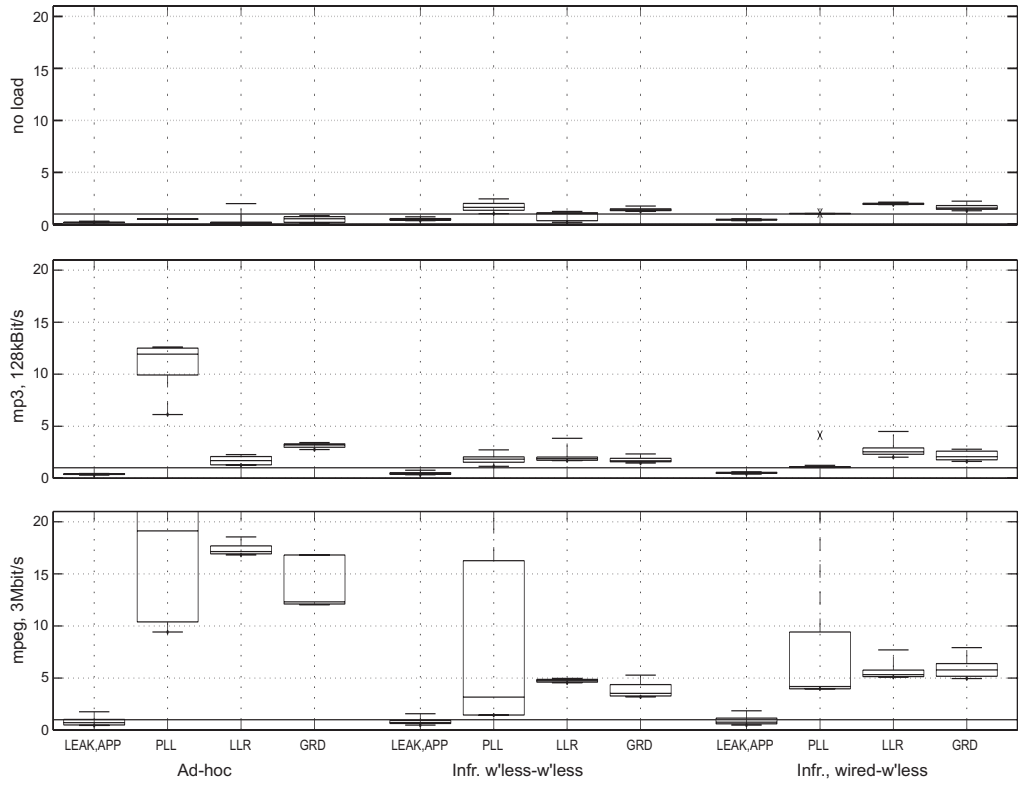


Fig. 29: Box plots of the performance penalty P achieved on ten traces per scenario and load type. In contrast to Fig. 28, a single parameterization is used for all scenarios and load types. The median and maximal penalties of the algorithms used in this figure are summarized in Tab. 9.

What is the achieved performance?

Clearly, it is desirable to have a CSA that works well in *every* scenario and under *variable* network load without changing its parameters. Thus we examine the performance of the CSAs with a single parameterization in different scenarios. Figure 29 shows the achieved performance penalty P of a single algorithm optimized for all scenarios concurrently. For every scenario and load type, ten traces were evaluated and the resulting penalties are displayed using box plots[6], the median and maximal values are summarized in Tab. 9. The penalties are slightly larger than in the last experiment, shown in Fig. 28, because the CSAs' parameters are not optimized for every scenario individually. Algorithm $\mathcal{A}^{\text{pll}}$ seems to be the least robust of the examined CSAs, in the sense that its performance penalty under heavy network load is more widely distributed than that of the other algorithms.

[6]Box plots show maximum, minimum (tails) and median values, as well as upper and lower quartiles (box) of a distribution.

Ad-hoc	*Median penalty*			*Maximal penalty*		
	-	128kb/s	3Mb/s	-	128kb/s	3Mb/s
$\mathcal{A}^{\text{leak,app,adap}}$	0.20	0.38	0.72	0.56	0.59	1.98
$\mathcal{A}^{\text{pll}}$	0.52	11.92	19.12	0.69	13.2	> 20
$\mathcal{A}^{\text{llr}}$	0.16	1.69	17.14	2.01	2.40	18.62
$\mathcal{A}^{\text{grd}}$	0.56	3.20	12.30	0.98	3.76	17.17
Infr. w'less-w'less	*Median penalty*			*Maximal penalty*		
	-	128kb/s	3Mb/s	-	128kb/s	3Mb/s
$\mathcal{A}^{\text{leak,app,adap}}$	0.62	0.62	0.89	0.87	0.96	1.81
$\mathcal{A}^{\text{pll}}$	1.83	1.85	4.33	2.74	3.21	> 20
$\mathcal{A}^{\text{llr}}$	1.02	1.98	4.90	1.18	4.64	5.03
$\mathcal{A}^{\text{grd}}$	1.20	1.77	4.15	1.42	2.71	5.16
Infr. wired-w'less	*Median penalty*			*Maximal penalty*		
	-	128kb/s	3Mb/s	-	128kb/s	3Mb/s
$\mathcal{A}^{\text{leak,app,adap}}$	0.46	0.52	0.76	0.71	0.92	2.05
$\mathcal{A}^{\text{pll}}$	1.07	1.05	4.19	1.13	4.78	> 20
$\mathcal{A}^{\text{llr}}$	1.96	2.54	5.33	2.17	4.81	8.07
$\mathcal{A}^{\text{grd}}$	1.59	2.07	5.78	2.22	3.26	8.10

Tab. 9: Median and maximal performance penalties P of the algorithms used for Fig. 29. $P \leq 1$ means that all requirements are met.

How much does it cost?

Finally, we compare the cost of the CSAs in terms of computation and memory overhead. The results are shown in Tab. 10. Algorithms $\mathcal{A}^{\text{pll}}$ and $\mathcal{A}^{\text{grd}}$ are cheapest, algorithm $\mathcal{A}^{\text{leak,app,adap}}$ is slightly more expensive, and algorithm $\mathcal{A}^{\text{llr}}$ is much more expensive, since it uses much more memory than the other CSAs. However, this result can be misleading, since cost was not an optimization criterion. Our experience shows that algorithm $\mathcal{A}^{\text{llr}}$ with a window size of approximately 100 time stamps achieves almost equal results as the one generated by the evolutionary optimization with a window size of more than 40'000. Still, algorithm $\mathcal{A}^{\text{grd}}$ achieves equal and even better results than $\mathcal{A}^{\text{llr}}$ at a much lower cost.

3.6.3 Discussion

In the first experiment we have seen that for algorithms $\mathcal{A}^{\text{pll}}$, $\mathcal{A}^{\text{llr}}$, and $\mathcal{A}^{\text{grd}}$, the peak jitter requirement is difficult to achieve. In the second experiment, it was shown that this is mostly a problem in heavy-network-load situations. A possible explanation for this phenomenon is that all these algorithms in some way average over many time stamps. The average and median message delay is strongly dependent on the network load (see Tab. 3). The synchronization error achieved by these algorithms mirrors the short-term variations of the median message delay. Figures 28 and 29 indicate that the ad-hoc scenario with high load is most difficult for the algorithms $\mathcal{A}^{\text{pll}}$, $\mathcal{A}^{\text{llr}}$, and $\mathcal{A}^{\text{grd}}$. This matches with

Algorithm	*Operations*				*Memory*
	n^+	$n^<$	n^*	$n^/$	
$\mathcal{A}^{\text{leak,app,adap}}$	9	3	5	2	32
$\mathcal{A}^{\text{pll}}$	8	3	5	0	7
$\mathcal{A}^{\text{llr}}$	15	4	10	2	43996
$\mathcal{A}^{\text{grd}}$	13	4	4	5	8

Tab. 10: Resource requirements of the algorithms used for the results from Fig. 29. The table shows the number of operations sorted by the type of operation ($n^<$ is the number of comparisons, $n^/$ also includes modulo operations) and the number of time stamps that are stored in memory. In our implementation, time stamps are 8 bytes long.

the variability of the median delay under network load, shown in Tab. 3. While the median delay in the ad-hoc scenario varies by more than 600μs, it varies by less than 200μs in the infrastructure scenarios.

In contrast, algorithm $\mathcal{A}^{\text{leak,app,adap}}$ achieves a far better peak jitter and this almost independently of the network-load situation. This is explained by the fact that in contrast to all other algorithms, $\mathcal{A}^{\text{leak,app,adap}}$ selects time-stamp messages with a delay close to the *minimal* delay. In contrast to the average and also the median delay, the minimal delay remains constant even under heavy-network-load conditions, as can be verified in Tab. 3. Algorithms $\mathcal{A}^{\text{leak,app,adap}}$ has more troubles with the MTIE criterion. The reason may be the selective nature of the algorithm, resulting in frequent "jumps" of the synchronization error. We have seen in the experimental study of Sect. 3.5.5 that drift compensation and parameter adaptation reduce this problem.

Considering the median performance penalties shown in Tab. 9, only algorithm $\mathcal{A}^{\text{leak,app,adap}}$ achieves all target metrics in all scenarios and with all load types, the other algorithms have $P > 4$ in the infrastructure scenarios and even $P > 12$ in the ad-hoc scenario. Considering the maximal performance penalties, algorithm $\mathcal{A}^{\text{leak,app,adap}}$ misses the targets by a factor of 2, which may still be acceptable for many consumer applications. The penalties of algorithms $\mathcal{A}^{\text{llr}}$ and $\mathcal{A}^{\text{grd}}$ are more than twice as large ($P > 5$) in the infrastructure scenarios and more than eight times larger in the ad-hoc scenario. In all scenarios, algorithm $\mathcal{A}^{\text{pll}}$ performs much worse than the other algorithms. This confirms the finding of Noro's thesis ([Nor00]), which claims that linear regression often achieves a better synchronization than PLL algorithms.

In summary, we have shown that for wireless loudspeakers, $\mathcal{A}^{\text{leak,app,adap}}$ performs considerably better than the other algorithms and mostly achieves the target metrics. The memory and computation overhead of all algorithms is small, which allows embedded-system implementations at driver level or even in hardware.

3.7 Summary and Conclusions

In this chapter, we have studied time synchronization of a single client node that receives time-stamp messages from a reference node. We have presented the classes of Local-Selection (LS) and heuristic Local-Selection (HLS) CSAs, an overview is provided in Figure 30. Several ways to analyze these algorithms have been presented. Experimental evidence has been provided that these novel algorithms substantially outperform previously known state-of-the-art CSAs. In the following, we summarize the chapter. Important conclusions are *emphasized*.

3.7.1 Analysis

The LS-CSAs have been analyzed under various assumptions about message delays and clock drift. In Sect. 3.2, it has been shown that LS-CSAs are safe and live. These properties assure that *without making assumptions about message delays, it is guaranteed that the synchronization error does not grow boundlessly*. In addition, LS-CSAs fulfills the stronger optimally-selective property, which formally states that these algorithms never make a wrong decision.

The optimally-selective property is used in Sect. 3.3 to derive a novel upper bound on the achieved peak jitter under the assumption of message delays constrained by an upper delay-interval curve. It has been shown that this assumption can provide *more detailed and more realistic estimations of the actually achieved peak jitter* than a previously known analysis based on upper- and lower-bounded message delays.

In Sect. 3.4, we have presented ways to estimate and compensate the drift of the client's local clock under the assumption that the CSA has some a-priori knowledge about message delays. It has been shown that *drift compensation substantially improves time synchronization*. We have found that variability of the clock drift makes it impossible to compensate drift completely. A simple procedure for estimating the achievable rate error has been proposed. The upper bound on the peak jitter presented in Sect. 3.3 has been refined to allow for the drift compensation mechanism of algorithm $\mathcal{A}^{\mathrm{lsdc,iter}}$.

In Sect. 3.5, it has been shown that *without a-priori knowledge about message delays, drift compensation is not possible* in a deterministic sense, i.e. every drift-compensation procedure risks to make an arbitrarily large error.

In summary, we have contributed to a better understanding of unidirectional point-to-point synchronization. We have clarified the relation between assumptions about the system and guarantees about the synchronization quality. The most eminent result is that the assumption of message delays constrained by an upper delay-interval curve allows to derive tight and guaranteed bounds on the synchronization quality. As we have shown in the last chapter, such delay-interval curves are quite simple to obtain. Thus, *guaranteed synchronization in the range of microseconds is possible in 802.11b WLANs.*

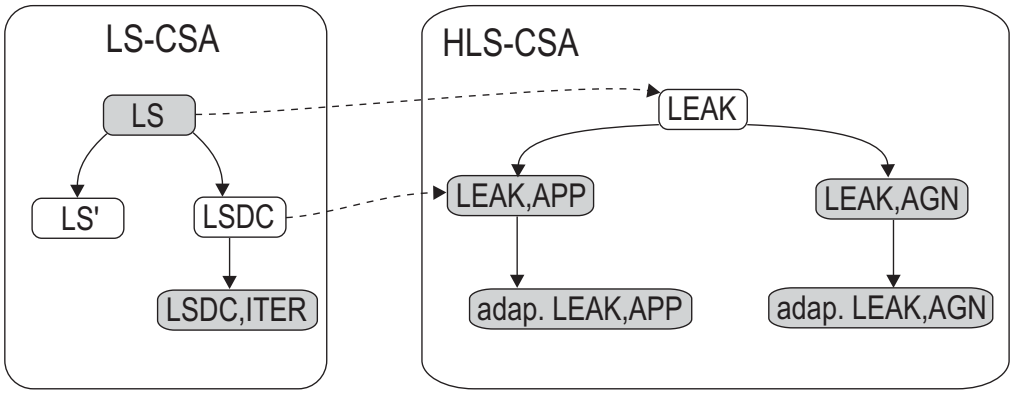


Fig. 30: Summary of the Local-Selection algorithms (LS-CSAs) and heuristics (HLS-CSAs) presented in Chapter 3. The algorithms in shaded boxes are completely specified and implementable. A solid arrow connects a basic algorithm with a refined variant. Dashed arrows represent a more general similarity between two algorithms.

3.7.2 Practical Algorithms

A second goal of this chapter was to present *simple algorithms that require minimal a-priori knowledge about the system* and which can be used in a wide range of system setups and application scenarios without making adaptations.

In Sect. 3.5, the heuristic Local-Selection algorithms have been presented. They are safe and live, thus the synchronization error does not grow boundlessly. In contrast to LS-CSAs, HLS-CSAs are not optimally selective and thus the upper bounds developed in Sect. 3.3 do not hold. Instead, their performance has been assessed experimentally. It has been shown that *HLS-CSAs perform substantially better than other CSAs* if, e.g. due to heavy and variable network load, the median message delay is variable. It has been shown that *HLS-CSAs achieve the synchronization quality required for wireless loudspeakers*. This result has been confirmed by our industrial partner BridgeCo[7], who has implemented HLS-CSAs in commercial products. BridgeCo holds an international patent on the fundamental slow-logical-clock property of LS-CSAs and the eventually-slow-logical-clock property of HLS-CSAs [BD01].

3.7.3 Outlook

In the next chapter, we present interval-based synchronization, which is concerned with providing a client node with upper and lower bounds on reference time instead of with a single estimate of reference time. In Sect. 3.2, we have shown that LS-CSAs actually compute a lower bound on reference time (Lem. 3). We will thus show in Chapter 5 how *LS-CSAs can be used to construct interval-based CSAs*.

[7] BridgeCo AG, 8600 Dübendorf, Switzerland. *http://www.bridgeco.net*

4

Multihop Synchronization

In Sect. 1.1, we have identified three fundamentally different issues in clock synchronization: (i) Message delay between synchronizing nodes, (ii) clock drift and (iii) the organization of synchronization in complex arrangements of many nodes. In the last chapter, we have extensively discussed the first two issues in a very simple scenario of a single client node synchronizing to a single reference node. The purpose of this chapter is to discuss the third issue, namely that of organizing clock synchronization in large multihop networks, which are typical for example in sensor-network applications.

In Sect. 4.1, the specific problems arising in multihop synchronization are explained and known approaches to solve them are presented. A formal system model is introduced and a lower bound for the worst-case performance is derived.

In Sect. 4.2, we present *interval-based* synchronization as a novel multihop synchronization strategy. It is cheaper than the previously known clustering and tree-based schemes, since it *does not require any coordination* among the nodes to setup or maintain global configuration of the synchronization process. As a consequence, the scheme is more robust against changes in the network topology than other schemes, i.e., against nodes that leave or enter the system, mobile nodes, link failures, etc. Furthermore, it is shown that the interval-based algorithm $\mathcal{I}^{\mathrm{IM}}$ from Marzullo and Owicki [MO83] is worst-case optimal in arbitrary scenarios, while tree-based schemes are not. It is shown that interval-based algorithms can simultaneously achieve a good performance and a balanced distribution of the energy consumption among all nodes.

In Sect. 4.3, the interval-based algorithm $\mathcal{I}^{\mathrm{BP}}$ is presented, which outperforms algorithm $\mathcal{I}^{\mathrm{IM}}$ in typical cases and achieves the same performance in the worst case. These algorithms are compared by simulation of typical scenarios.

4.1 Introduction

In this section, we first describe informally why multihop synchronization is a more difficult problem than point-to-point synchronization (Sect. 4.1.1). A short discussion summarizes currently known approaches to cope with these difficulties (Sect. 4.1.2). In Sect. 4.1.3, the system model is formalized. Finally, a lower bound on the worst-case accuracy is derived (Sect. 4.1.4).

4.1.1 The Problem

In *multihop networks*, nodes cannot communicate with all other nodes in the system directly. The emerging application of large-scale sensor networks is one example of a multihop network. Sensor nodes typically are small and inexpensive devices with stringent energy constraints. As the power consumption for communication increases quickly with the geographical distance between sender and receiver, multihop communication is potentially more energy-efficient than direct links between all sensor nodes.

Concerning clock synchronization, multihop communication creates a problem: As not all nodes can directly communicate with the reference node, two client nodes N_u and N_v have to synchronize directly with each other. But what does this mean? Should N_u set its time according to that of N_v or should rather N_v adjust its clock to N_u's? Clearly it is desirable that the node whose clock has a smaller error is used as the reference for the other. But how do the nodes know which one that is? In Sect. 4.1.2, state-of-the-art solutions to this problem are discussed.

There are also other issues in multihop synchronization. multihop networks are sometimes deployed in an *ad-hoc* fashion. Consider the example of environmental monitoring using a large number of sensor nodes. Some of these nodes may go out of service because their batteries are exhausted. New nodes may be added to the system to replace the failed nodes. Nodes may be *mobile*, for example if attached to wild animals, to weather balloons, or to soldiers in a battlefield. Ad-hoc deployment and mobility cause permanent change in the system. It is thus desirable that synchronization schemes adapt to this change quickly and without causing a large overhead in terms of communication.

4.1.2 Related Work

In this section, we present two ways to avoid the multihop synchronization problem sketched in the previous section, and two ways to explicitly deal with the issue. Figure 31 illustrates these approaches.

Virtual connections
An overlay network of virtual connections from all nodes to a reference node can be constructed. Synchronization over these virtual connections then is reduced to the simple reference-client situation discussed extensively in Chap. 3. The approach has some major drawbacks: (i) The message delay on the virtual

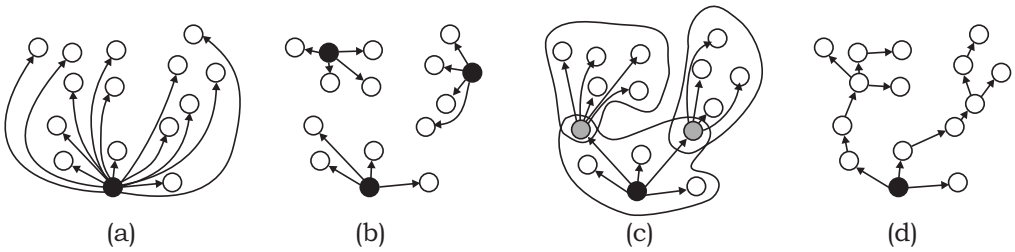


Fig. 31: *Organizing synchronization in multihop networks.* (a) Virtual connections from the reference to all clients. (b) Single-hop synchronization with reference nodes that are synchronized out-of-band (e.g., via GPS). (c) Single-hop synchronization in overlapping clusters, gateway-nodes translate (grey) time stamps. (d) Tree hierarchy with a single reference node at the root.

connections can become extremely large and widely variable, since messages are repeatedly received, stored and forwarded. The situation is aggravated if nodes are temporarily switched off in order to reduce energy consumption. (ii) It may be quite difficult and expensive to maintain the virtual connections if the nodes are mobile. (iii) In networks with a large number of nodes, the central reference node becomes a bottleneck; the approach does not scale.

Out-of-Band Synchronization

Another way of circumventing the problem is to deploy a large number of reference nodes in the network, such that every node has a direct connection to one of them. The reference nodes are synchronized among each other using some out-of-band mechanism. The global positioning system (GPS) is well suited for this purpose as it provides time information with sub-microsecond accuracy. Such a system has been proposed by Verissimo et al. [VRC97]. However, GPS receivers are still relatively costly. Additionally, they require a direct line of sight to a number of satellites, and thus cannot operate inside buildings.

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 every node in the cluster maintains multiple logical clocks that are synchronized with all other nodes in the cluster. Some nodes are members in several clusters and participate independently in all corresponding synchronization procedures. These nodes act as time gateways and translate time stamps from one cluster to the other. Clustering has been proposed for internal synchronization, that is the synchronization of a subset of client nodes. For external synchronization to an a-priori designated reference node, the clustering scheme is equivalent to tree construction discussed below; every cluster corresponds to one hop in the tree, see Fig. 31.

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 improves the accuracy; on the other hand, energy consumption grows quickly with increasing transmission range, which is in favor of many small clusters. This trade-off has been examined in [MR03].

The disadvantages of the clustering approach, at least in the form proposed by [EGE02], are that (i) a broadcast-communication mechanism is required, and (ii) the construction and maintenance of the cluster organization becomes difficult in networks with mobility and nodes leaving and joining.

Tree Construction

The most common solution of the multihop synchronization problem is to construct a synchronization tree with a single reference at the root, see [GKS03, SV03, vGR03, MKSL04a]. Single-hop synchronization is applied along the edges of the tree with the node closer to the root as the reference node. Various well-known algorithms can be used to construct such a tree, a discussion of this is presented in [vGR03]. As the accuracy degrades with the hop distance from the root, a tree with minimum depth is preferable. Tree construction faces two main problems: (i) In sensor networks, the network topology is dynamic; nodes temporarily or permanently fail and thus leave and rejoin the network. The tree-based synchronization algorithms explicitly have to deal with such events. In particular, the root node may fail, which necessitates to elect a new root [MKSL04a]. (ii) Two neighboring nodes in terms of physical location may have a large hop distance in the synchronization tree. In consequence, the accuracy of synchronization between these nodes is not as good as if they would synchronize with each other directly.

4.1.3 Model and Problem Statement

We now formally define the multihop synchronization problem. From this problem definition, a lower bound on the achievable worst-case performance is derived.

Def. 24: **(Network and Nodes)** *A network $\mathcal{N} = \{N_u | 0 \leq u \leq n\}$ is a set of $n+1$ nodes. Every node N_u has a local clock H^u with a drift rate ρ^u in the range of $[-\hat{\rho}, \hat{\rho}]$ and an arbitrary offset relative to real time. The node N_0 is the reference node and its local clock shows real time, that is $H^0(t) = t$. All other nodes are referred to as client nodes.*

Def. 25: **(Communication Events)** *Communication is point-to-point between two nodes. Communication is modeled as an event and thus has no duration; every communication event occurs at a unique real time. At a communication event, the nodes can exchange an arbitrary amount of information in both directions. Formally, the i-th communication event is a tuple (u_i, v_i, t_i), meaning that nodes N_{u_i} and N_{v_i} communicate at real time t_i.*

Note that this communication model assumes zero delay, which is completely different from the models used in the previous chapters.

Def. 26: **(Scenario)** *A scenario $\mathcal{S} = (\mathcal{N}, \{(u_i, v_i, t_i)\})$ is a graph with vertices that correspond to nodes in the network $\mathcal{N}$ and edges (N_{u_i}, N_{v_i}) with weight t_i that correspond to communication events.*

Figure 32 shows two representations of a scenario. On the left side, the graph representation is shown. On the right, the scenario is represented as an event chart: For every client node, a time line is drawn. Communication events with the reference node are shown as bullets on the time line of the corresponding client node at the height corresponding to the real time of this event. Communication events between two client nodes are shown as a connected bullet pair on the time lines of the involved nodes, again at the height corresponding to the real time of the communication event.

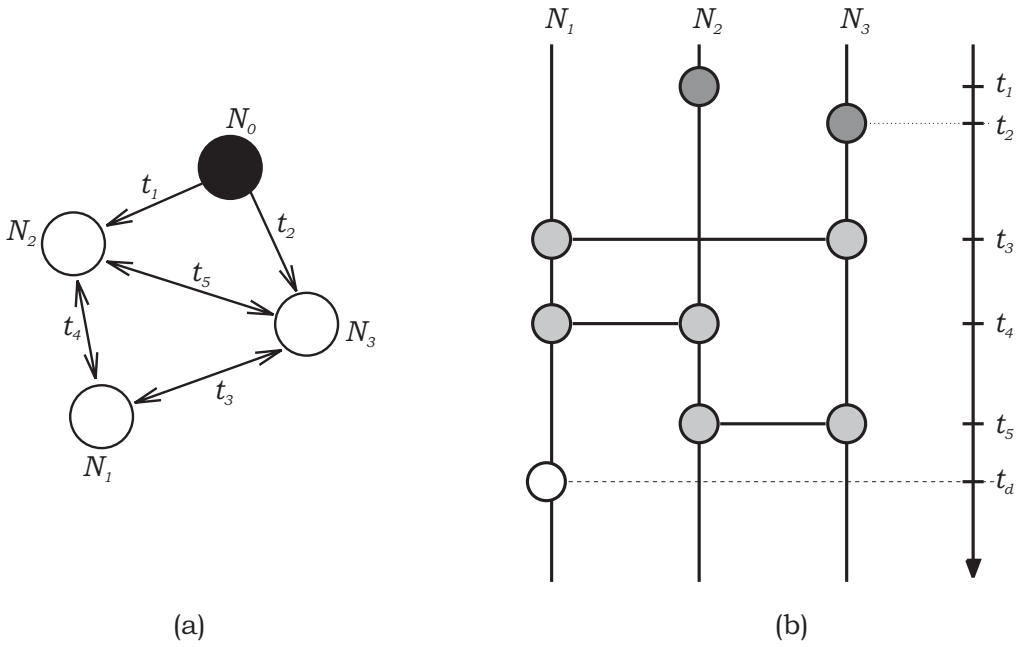


Fig. 32: Graph and event-chart representations of a scenario.

A scenario is a form of specifying a particular synchronization problem. We will analyze the following: Given a scenario, what is the accuracy a particular CSA can guarantee?

To answer this question, it has to be specified precisely what we understand by accuracy. By Def. 11, the accuracy is the largest absolute value of the synchronization error in a specified observation interval. We now take another approach:

Def. 27: **(Destination Event)** *A destination event is an artificial event used for the analysis of a scenario. It is modeled as a tuple (d, t_d), meaning that the synchronization error of node $N_d \in \mathcal{N}$ at real time t_d is analyzed.*

Clearly, the choice of the destination event has a strong influence on the result of the analysis. For example, if it is placed on a node that has never communicated with the reference node directly or via other client nodes, then no finite accuracy can be guaranteed. Also, if the destination event is placed long after the last communication event of the node N_d, then the worst-case accuracy will be very large. So how should the destination event be placed? There are three different interpretations of this problem: (i) Maybe the application that uses the synchronized time specifies itself when and at which node and at what time synchronization is needed. This approach will be followed in the next section where a lower bound for the achievable accuracy is derived. (ii) If a scenario consists of a periodic pattern, then it maybe immediately clear for which position of the destination event the accuracy becomes maximal. This approach will be followed in Sect. 4.2.2, where this lower bound is applied to scenarios corresponding to synchronization trees. (iii) In more complicated scenarios, the worst-case accuracy of many different destination events can be evaluated and the worst case of all these results can be determined by taking the maximum. This approach will be followed in Sect. 4.2.3.

4.1.4 Lower Bound

In this section, we derive a lower bound A_d for the accuracy at a destination event (d, t_d) achieved by a deterministic CSA in a given scenario $\mathcal{S}$. To this purpose, we have to complement the notion of a scenario by the notions of traces and views.

Def. 28: **(Trace)** *A trace $\mathcal{T} = (\mathcal{N}, \{(u_i, v_i, t_i, h_i^{u_i}, h_i^{v_i})\}$ is a scenario $\mathcal{S}$ augmented by the local times $h_i^{u_i}$ and $h_i^{v_i}$ of the nodes N_{u_i} and N_{v_i} involved in the i-th communication event.*

A trace is thus a concrete realization of a scenario. There are infinitely many traces corresponding to a scenario. These traces differ in the drift rates of the local clocks. By Def. 24, the drift rates are bounded and thus for every two communication events i and j where a node N_u is involved, the local time difference $h_j^u - h_i^u$ must lie in the interval $[(t_j - t_i)(1 - \hat{\rho}), (t_j - t_i)(1 + \hat{\rho})]$.

Like a scenario, a trace can be represented as a graph or as an event chart, see for example Fig. 33

While a trace contains the complete information about the real and local times of all communication events in a scenario, we now introduce the view, which contains the information that is known to a node at a particular time.

Def. 29: **(View)** *A view $\mathcal{V}_d$ is a set of tuples $(u_i, v_i, h_i^{u_i}, h_i^{v_i})$, containing all information about communication events, i.e. the involved node identifiers u_i and v_i and the respective local times $h_i^{u_i}$ and $h_i^{v_i}$, that is available to node N_d at real time t_d.*

Informally, it is simple to understand about which communication events node N_d can have information at real time t_d: the view $\mathcal{V}_d$ contains information

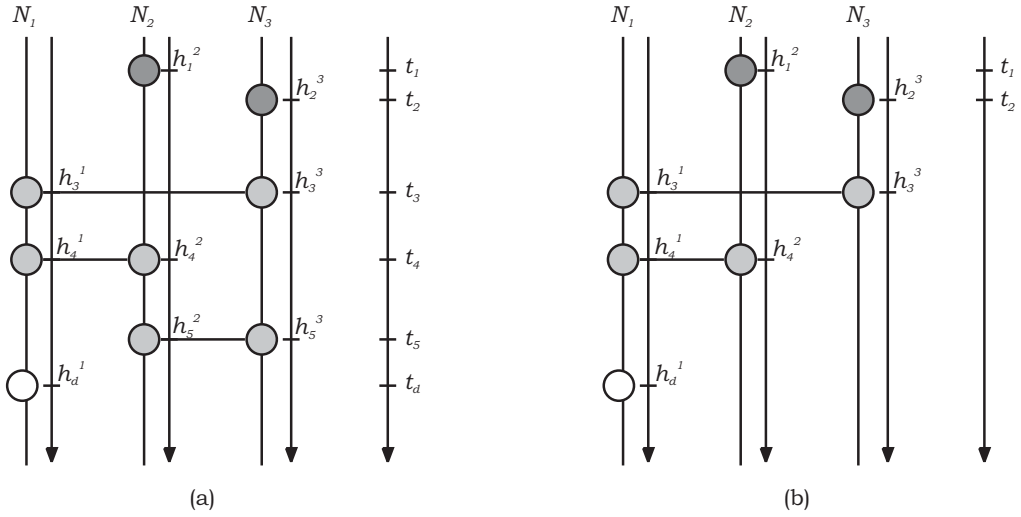


Fig. 33: Trace and view of the scenario from Fig. 32. The trace complete specifies real and local times of all events in the system. The view only contains the information available to a client node at a particular destination event.

about all communication events from which there exists a path to the destination event d in the event chart, without going backwards on the time lines. In other words, the view contains the information that node N_d has acquired locally, i.e. local times h^d of communication events in which it is involved, and the information it can have received from other nodes at a communication event.

Figure 33 shows on the left side a trace and a destination event. On the right side, the corresponding view is shown. In contrast to the trace on the left, the view on the right does not contain the communication event 3, since node N_1 cannot know about it at real time t_d. The view also does not contain the real times of the communication events among client nodes. It does, however, contain the real times of the communication events with the reference node, since the reference node knows the real time.

There are infinitely many traces that can correspond to a view. The traces differ in the drift rates of the nodes and thus the real times of communication events. As drift rates are bounded, the real time difference $t_j - t_i$ between any two communication events i and j in which node N_u is involved must lie in the interval $[(h_j^u - h_i^u)/(1+\hat{\rho}), (h_j^u - h_i^u)/(1-\hat{\rho})]$.

Thm. 17: **(Lower bound)** *Given a scenario $\mathcal{S}$ and a destination event (d, t_d), then the accuracy A_d of the synchronized time c_d computed by a deterministic CSA on node N_d at real time t_d is lower-bounded by*

$$A_d \geq \frac{\hat{\rho}}{1-\hat{\rho}} (t_d - t_s) \tag{4.1}$$

where t_s is the largest real time of any communication event with the reference node that is contained in the view $\mathcal{V}_d$.

Proof: The idea of the proof for Theorem 17 is the following: Given a scenario with a destination event d occurring at real time t_d, we construct a trace $\mathcal{T}$ and a view $\mathcal{V}_d$ such that all client's drift rates are maximal, i.e., $\rho^u = \hat{\rho}$ for all nodes N_u in the network. Let us assume that some deterministic CSA computes the estimated real time c_d from the view $\mathcal{V}_d$. Now we construct a second trace $\mathcal{T}'$, which has exactly the same view $\mathcal{V}_d$ as the first trace $\mathcal{T}$. But in $\mathcal{T}'$, the drift rates for all nodes N_u in the network are switched to $\rho^u = -\hat{\rho}$ after the latest communication event with the reference node has occurred. Since all local times are the same in both traces, the real times of all communication events occurring after the latest communication event with the reference node are increased, thus $\mathcal{T}'$ corresponds to a scenario $\mathcal{S}'$ that is different from $\mathcal{S}$. The two alternative traces having the same view are illustrated in Fig. 34.

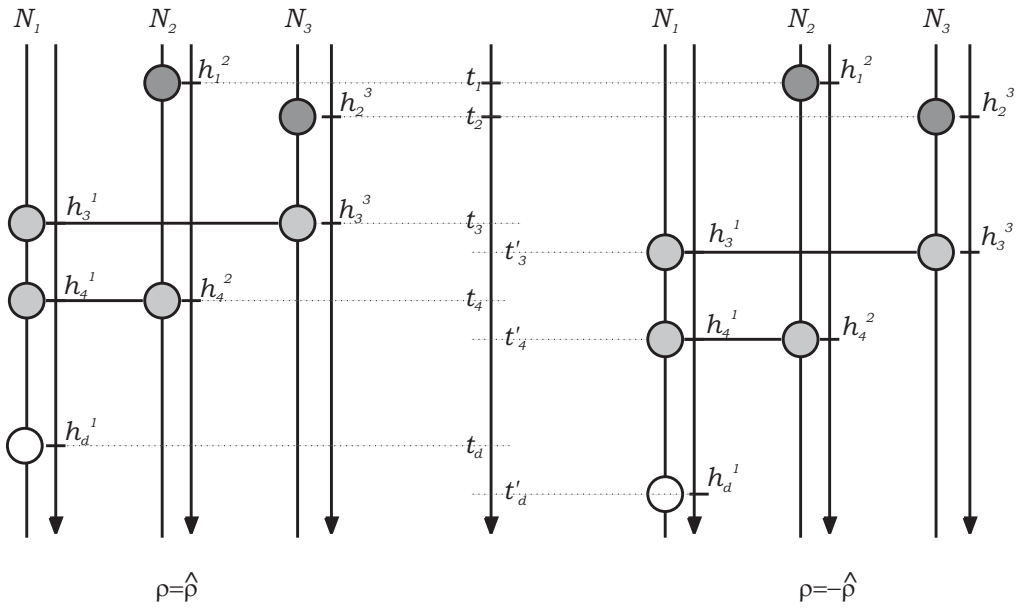


Fig. 34: Two traces with the same view. On the left, $\rho = \hat{\rho}$ at all times. After time t_2, the clocks in the constructed trace on the right run with minimal speed. Since the two traces are indistinguishable to a CSA, i.e. they have the same view, the accuracy achieved by a deterministic CSA cannot be better than $\frac{1}{2}(t'_d - t_d)$.

The CSA, as it is deterministic, thus computes the same estimated real time c_d for the destination event. But in the trace $\mathcal{T}'$, the real time of the destination event d is t'_d. The accuracy the CSA can guarantee is thus at best $\frac{1}{2}(t'_d - t_d)$ if $c_d = \frac{1}{2}(t_d + t'_d)$. In the following, it is shown that this expression is exactly the one from Thm. 17. Let Δh be the sum of all local time differences on an arbitrary path in the event chart from the latest communication event with the reference node to the destination event. Δh is equal for all paths, since we have assumed that all nodes have the same drift. For trace $\mathcal{T}$, we have $\Delta h = (1 + \hat{\rho})(t_d - t_s)$. For trace $\mathcal{T}'$, we have $\Delta h = (1 - \hat{\rho})(t'_d - t_s)$. Since the view of both traces is the same, Δh must be equal for both cases, thus we

have $(1+\hat{\rho})(t_d - t_s) = (1-\hat{\rho})(t'_d - t_s)$. This equation can be transformed to $t'_d = t_s + (t_d - t_s)\frac{1+\hat{\rho}}{1-\hat{\rho}}$ and thus $t'_d - t_d = (t_d - t_s)(\frac{1+\hat{\rho}}{1-\hat{\rho}} - 1) = (t_d - t_s)(\frac{2\hat{\rho}}{1-\hat{\rho}})$. Thus the accuracy can at best be $A_d = \frac{\hat{\rho}}{1-\hat{\rho}}(t_d - t_s)$. □

Ex. 12: **(Node Chain)** *Assume a network of nodes $\mathcal{N} = \{N_u | 0 \leq u \leq n\}$ that are arranged in a chain by ascending node identifier. Every pair of neighboring nodes communicates at least once in every interval of real-time length Δt. Fig. 35 illustrates the corresponding worst-case scenario. Worst-case here means that node N_u communicates with N_{u+1} immediately before N_u communicates with N_{u-1} thus information received by N_u from N_{u-1} can only be passed to N_{u+1} after an interval of length Δt.*

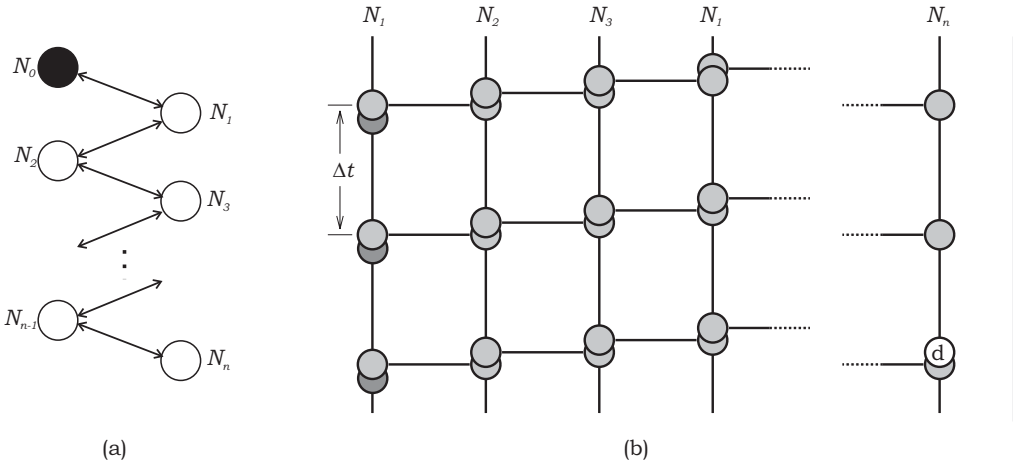


Fig. 35: Worst-case scenario and destination event for a chain of nodes

The real-time difference between the destination event and the latest communication event with the reference node in the corresponding view is $t_d - t_s = n\Delta t$. Using Eq. 4.1 we get a lower bound for the worst-case accuracy of node n of $A_d \geq \frac{\hat{\rho}}{1-\hat{\rho}} n\Delta t$. The lower bound A_d is thus proportional to the interval of communication events Δt and the hop distance to the reference node.

4.2 Interval-Based Synchronization

In this section, we propose interval-based synchronization as a novel approach to the multihop synchronization problem. In Sect. 4.2.2, we show that the simple interval-based algorithm $\mathcal{I}^{\mathrm{IM}}$, which has been presented in [MO83] for the single-hop case, always meets the lower bound from Theorem 17 and is thus worst-case-optimal. We then compare the worst-case accuracy of algorithm $\mathcal{I}^{\mathrm{IM}}$ with that of tree-based algorithms. For *arbitrary* scenarios, tree-based algorithms are not worst-case-optimal. In Sect. 4.2.3, scenarios that are *specifically*

designed for synchronization are considered. For a given frequency of communication events in a network, which corresponds to a given amount of energy per time spent on synchronization, algorithm $\mathcal{I}^{\mathrm{IM}}$ combines the advantages of different tree-construction principles, that is a good accuracy and an equal distribution of communication events, and thus energy consumption, among all nodes in the network.

4.2.1 Definitions

As in the case of estimate-based CSAs (see Def. 4), interval-based CSAs are executed locally on one node immediately after a communication event. We use the notation $\mathcal{V}_i^u$ for the view $\mathcal{V}_d$ according to Def. 29 with a destination event placed at node N_u immediately after the i-th communication event. In contrast to estimate-based CSAs, interval-based CSAs compute two clocks, a lower-bound clock $\check{C}$ and an upper-bound clock $\hat{C}$.

Def. 30: **(Interval-based CSA)** *An interval-based CSA $\mathcal{I}$ running on node N_u computes after every communication event i in which N_u is involved from the corresponding view $\mathcal{V}_i^u$ a new pair of logical clocks $(\check{C}_i^u, \hat{C}_i^u) = \mathcal{I}(\mathcal{V}_i^u)$. The lower-bound clock $\check{C}^u$ and the upper-bound clock $\hat{C}^u$ are defined as follows:*

$$\begin{aligned} \check{C}^u(h^u) &= \check{C}_i^u(h^u), \text{with } i : h_i^u \leq h^u < h_{i+1}^u, \\ \hat{C}^u(h^u) &= \hat{C}_i^u(h^u), \text{with } i : h_i^u \leq h^u < h_{i+1}^u \ . \end{aligned}$$

The goal of an interval-based CSA is to ensure that at all times, real time is above the reading of the lower-bound clock and below the reading of the upper-bound clock.

Def. 31: **(Correctness)** *An interval-based CSA $\mathcal{I}$ is correct if real time t is contained in the interval defined by the lower- and the upper-bound clocks, that is if*

$$\check{C}^u(H^u(t)) \leq t \leq \hat{C}^u(H^u(t)) \ .$$

Secondly, the interval between the two clocks should be as small as possible.

Def. 32: **(Uncertainty)** *The uncertainty U^u provided by an interval-based CSA is defined as*

$$U^u(t) = \hat{C}^u(H^u(t)) - \check{C}^u(H^u(h)) \ .$$

Note that in contrast to the synchronization error $E(t)$ (see Def. 4), the uncertainty $U^u(t)$ can be computed by the client node N_u itself.

It is straightforward to derive an estimate-based CSA from an interval-based algorithm.

Def. 33: **(Estimate-Based CSA derived from an Interval-Based CSA)** *An estimate-based CSA $\mathcal{A}$ according to Def. 4 can be derived from any interval-based CSA $\mathcal{I}$ according to*

$$C(H(t)) = \frac{1}{2}(\check{C}(H(t)) + \hat{C}(H(t))) \ ,$$

where $\check{C}$ is a lower- and $\hat{C}$ an upper-bound clock computed by algorithm $\mathcal{I}$.

Deriving an estimate-based CSA from an interval-based CSA allows to evaluate the accuracy and other metrics. This is useful when comparing interval-based CSAs with estimate-based CSAs, which is done in the next sections.

4.2.2 A Worst-Case-Optimal Interval-Based Algorithm

The algorithm IM from [MO83] is an interval-based CSA. We present this algorithm here to illustrate the concept of interval-based CSAs and then show that it always achieves the lower-bound from Theorem 17.

Def. 34: (Algorithm IM) *In a network of nodes executing the interval-based algorithm $\mathcal{I}^{\mathrm{IM}}$, every node N_u computes after every communication event i with a node N_v lower- and upper-bound clocks according to*

$$\check{C}_i^u(h^u) = \begin{cases} 0 & \textit{if } i = 0 \\ \max(\check{c}_i^u, \check{c}_i^v) + \frac{h^u - h_i^u}{1+\hat{\rho}} & \textit{otherwise} \end{cases} \tag{4.2}$$

$$\hat{C}_i^u(h^u) = \begin{cases} \infty & \textit{if } i = 0 \\ \min(\hat{c}_i^u, \hat{c}_i^v) + \frac{h^u - h_i^u}{1-\hat{\rho}} & \textit{otherwise} \end{cases} \tag{4.3}$$

where the time stamps $\check{c}_i^u$ and $\hat{c}_i^u$ are the current value of the lower- and upper-bound clocks of node N_u immediately before the communication event, and $\check{c}_i^v$ and $\hat{c}_i^v$ are the corresponding time stamps of node N_v

At every communication event i between nodes N_u and N_v, the nodes exchange the current readings of their lower- and upper-bound clocks, that is, node N_u receives from N_v the time stamps $\check{c}_i^v$ and $\hat{c}_i^v$, and node N_v receives from N_u the time stamps $\check{c}_i^u$ and $\hat{c}_i^u$. The new lower-bound clock $\check{C}_i^u$ starts with the larger value of the two lower-bound clocks $\check{C}^u$ and $\check{C}^v$ at the time of the communication event i. It then progresses at a rate of $(1+\rho^u)/(1+\hat{\rho})$. The new upper-bound clock $\hat{C}_i^u$ starts with the smaller value of the upper-bound clocks $\hat{C}^u$ and $\hat{C}^v$ at the time of the communication event i. It then progresses at a rate of $(1+\rho^u)/(1-\hat{\rho})$.

Thm. 18: (Correctness of $\mathcal{I}^{\mathrm{IM}}$) *Algorithm IM is correct.*

Proof: To show that algorithm IM is correct, we have to show that (i) the initial lower- and upper-bound clocks are correct, (ii) that all later lower- and upper-bound clocks are initially correct, and (iii) that all lower- and upper-bound clocks remain correct. In the following, we prove these three points for the lower-bound clocks. The proof for the upper-bound clock can be derived analogously.

Clearly all clocks $\check{C}_0^u$ are correct, since real time is always positive, which proves (i). All clocks $\check{C}_i^u$ with $i > 0$ are initially equal to either $\check{C}_{i-1}^u$ or $\check{C}_{i-1}^v$, thus if $\check{C}_{i-1}^u$ and $\check{C}_{i-1}^v$ are correct, then also $\check{C}_i^u$ is initially correct. If $\check{C}_i^u$ is initially correct, then it remains correct, since its rate is $(1+\rho^u)/(1+\hat{\rho})$, which, by Def. 24, is never greater than 1. This proves (ii) and (iii). □

Thm. 19: (Upper bound on the Uncertainty of $\mathcal{I}^{\mathrm{IM}}$) *In a network of nodes executing algorithm IM, the uncertainty at a destination event d is upper-bounded by*

$$u_d \leq \frac{2\hat{\rho}}{1-\hat{\rho}}(t_d - t_s) \ ,$$

where t_s is the real time of the latest communication event with the reference node in the view $\mathcal{V}_d$.

Proof: The idea of the proof is the following: Select an arbitrary path in the event chart (only forwards on the time lines, both directions on the communication events) from the last communication event with the reference node in the view $\mathcal{V}_d$ to the destination event. Now we compute the uncertainty along this path. We assume that at all communication events on this path, the lower- and upper-bound clocks of the node from which the path starts the traversal of the communication event are better, i.e., the lower-bound clock has a larger value and the upper-bound clock has a smaller value, than the clocks of the node on which the path continues. The resulting uncertainty is an upper bound on the uncertainty achieved by algorithm IM.

The uncertainty at the start of the path is $u_s = 0$. On a path-segment along the time line corresponding to some node N_u, starting at communication event i and ending at communication event j, the uncertainty grows according to Def. 34 by

$$u_j^u - u_i^u = \frac{h_j^u - h_i^u}{1-\hat{\rho}} - \frac{h_j^u - h_i^u}{1+\hat{\rho}} = (h_j^u - h_i^u)\frac{2\hat{\rho}}{1-\hat{\rho}^2} \ .$$

By Def. 24, $(h_j^u - h_i^u) \leq (t_j - t_i)(1+\hat{\rho})$ and thus

$$u_j^u - u_i^u \leq (t_j - t_i)\frac{2\hat{\rho}}{1-\hat{\rho}} \ .$$

On the path segments corresponding to communication events, the uncertainty remains constant. Thus we can sum up the increase of the uncertainty over all path segments corresponding to time lines of some nodes N_u resulting in $u_d - u_s = (t_d - t_s)\frac{2\hat{\rho}}{1-\hat{\rho}}$, and since $u_s = 0$ we have proven Theorem 19. □

Thm. 20: (Worst-Case-Optimality of $\mathcal{A}^{\mathrm{IM}}$) *Algorithm $\mathcal{A}^{\mathrm{IM}}$ is worst-case-optimal for every scenario $\mathcal{S}$ and every destination event d.*

Proof: We show that the accuracy of algorithm $\mathcal{A}^{\mathrm{IM}}$ is half the uncertainty achieved by algorithm $\mathcal{I}^{\mathrm{IM}}$. By Theorems 17 and 19, this proves that $\mathcal{A}^{\mathrm{IM}}$ is worst-case-optimal.

As we know that $\mathcal{I}^{\mathrm{IM}}$ is correct, the synchronization error of the lower-bound clock $\check{C}$ is at most 0, and that of the upper-bound clock $\hat{C}$ is at most u_d. In this case, the error of the synchronized clock C is $u_d/2$. The synchronization error of the lower-bound clock is at least $-u_d$, and that of the upper-bound clock is at least 0. Then the error of the synchronized clock C is $-u_d/2$. In all other cases, the synchronization error is between these extremal values, and thus the accuracy $A_d = u_d/2$ of algorithm $\mathcal{A}^{\mathrm{IM}}$ is half the uncertainty of $\mathcal{I}^{\mathrm{IM}}$. □

Discussion

When we want to compare the interval-based approach with tree-based approaches, which have shortly been presented in Sect. 4.1.2, we face the difficulty that the interpretation of a scenario is different for the two approaches. In the interval-based approach, a scenario can be completely arbitrary. It is assumed that the scenario is generated by some functionality of the network that has no relation to synchronization, and algorithm $\mathcal{A}^{\mathrm{IM}}$ is worst-case-optimal in every scenario. In the tree-based approach, this is not so: synchronization information is only exchanged between parent and child nodes in the tree, thus a scenario cannot contain communication events between two nodes that are not neighbors in the tree.

One possibility of comparing the approaches is the following: We assume that an arbitrary scenario is given, since it was generated by some other functionality of the network. As stated before, algorithm $\mathcal{A}^{\mathrm{IM}}$ is worst-case-optimal. But what about the tree-based approaches? We assume the network is organized in some sort of tree, and every communication event in the scenario that is between nodes that are not neighbors in this tree is ignored by the synchronization algorithm.

Consider the example shown in Figs. 32, 33, and 34, and assume that the reference node N_0 is the parent of nodes N_2 and N_3, while node N_3 is the parent of N_1. Thus, the communication event between nodes N_2 and N_3 and the communication event between nodes N_2 and N_1 are not considered for synchronization. If we assume that there exists a tree-based CSA that achieves the lower bound of Theorem 17, then this algorithm is optimal for this scenario.

Now consider the same example, but assume that node N_2 is the parent of N_1. The same tree-based CSA, by Theorem 17, thus cannot achieve a better worst-case accuracy than $A_d \geq \frac{\hat{\rho}}{1-\hat{\rho}}(t_d - t_1)$ while the optimum is $A_d \geq \frac{\hat{\rho}}{1-\hat{\rho}}(t_d - t_2)$. Therefore, the tree-based CSA is not optimal. The interpretation of this example is that a tree-based CSA cannot make optimal use of some arbitrary scenario, since it is restricted to consider only synchronization-related information received from its parent.

4.2.3 Experimental Study: Comparison with Tree-Based Algorithms

We now present an approach for comparing interval-based CSAs with tree-based CSAs, assuming that scenarios are specifically designed for the respective synchronization principle. For the tree-based CSAs, this means that communication events only occur between corresponding parent and child nodes, thus also tree-based CSAs use all communication events, as interval-based CSAs do.

A tree can be seen as a set of chains, which are possibly overlapping. A network of n client nodes and a reference node can for example be organized in a single chain of length n or in n short chains with length 1 - a topology also called “star”. In Example 12, it has been derived that in a chain of length n, the

accuracy is lower-bounded by

$$A_d \geq \frac{\hat{\rho}}{1-\hat{\rho}} n\Delta t \tag{4.4}$$

if at least once in every interval of real-time length Δt a communication event occurs between every pair of neighbors. Thus if the network is organized in a single chain, the accuracy becomes rather bad; it increases proportionally with the length n. If the network is organized in n short chains, the accuracy is far better, that is $A_d \geq \frac{\hat{\rho}}{1-\hat{\rho}}\Delta t$, which is independent of the network size n. In both cases, the total number of communication events is the same, that is n in every interval of real-time length Δt. In the case of the long chain, these communication events are equally distributed among all nodes (two per node and per Δt, except for N_0 and N_n, which have only one in the same period). In the case of the short chains, all client nodes have one communication event per Δt and the reference node has n communication events per Δt. A single long chain has the advantage that all nodes communicate approximately equally often but have a bad accuracy. In contrast, many short chains provide a good accuracy, but the reference node has to serve many communication events, thus needs far more energy than the other nodes. It thus fails far earlier than the other nodes if all nodes have equal energy resources.

Clearly, both a single long chain and n short chains (star) are extreme cases of a tree. In the following, we explore the trade-off between accuracy and inequality of the number of communication events per time among the nodes in an experimental study. In this study, we also compare these metrics with the same metrics achieved by the interval-based approach.

Experimental setup

100 client nodes are randomly placed with a uniform probability distribution in a square are of width 100. In addition, a reference node is placed in the center of the square. With a given transmission range, a graph is constructed in which the nodes are the vertices and every pair of nodes with a distance below the transmission range is connected by an edge. In this graph, we construct two trees, one according to a depth-first algorithm, a second according to a breadth-first algorithm. These tree-based algorithms have been proposed for the use in synchronization algorithms by [vGR03]. Figure 36 illustrates this setup for three different transmission ranges.

For every generated tree, the maximal hop distance to the reference node and the maximal degree of any node are evaluated. Figure 37 shows these values for various transmission ranges from 10 to 70. For every value of the transmission range, 50 graphs have been generated, and the maximal and minimal values are shown in the figure.

To compare the tree-based approach with the interval-based approach, we proceed as follows: In the randomly generated networks, we let every node communicate with a randomly selected neighbor once in every interval with real-time length Δt. Thus, the total number of communication events per Δt

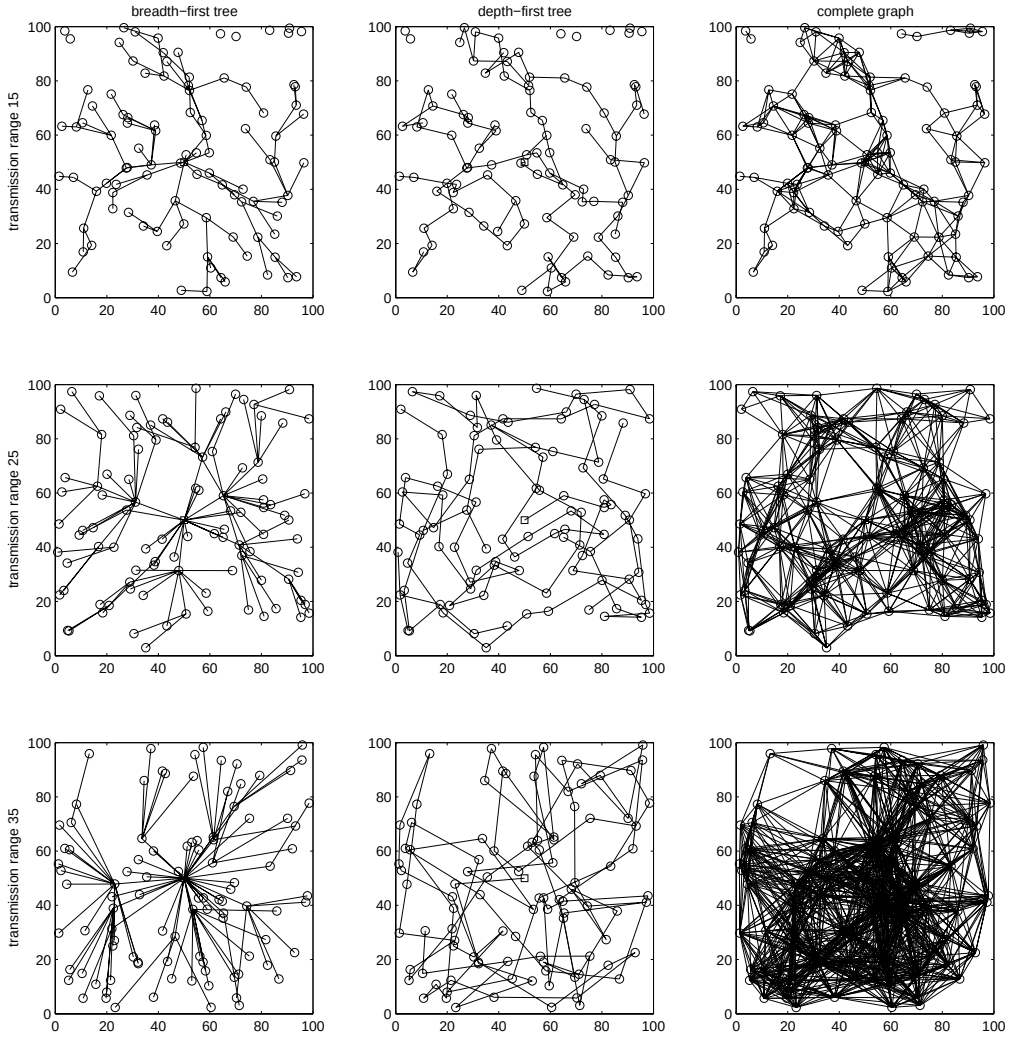


Fig. 36: Randomly generated networks for various transmission ranges. Left column shows a breadth-first tree, the middle column shows a depth-first tree and the right column shows the complete graph.

is the same as in the tree-based approaches. For every graph, this procedure is executed during a total time of $1000\Delta t$. A pseudo degree is computed by computing the total number of communication events per node divided by 1000 and taking the maximum. A pseudo hop distance to the root is computed as the maximal real-time length of any path in the scenario from a communication event with the reference node to any of the client nodes divided by Δt. The results are also shown in Fig. 37.

Results

First consider the maximal degree of any node in the network, shown on the right side of Fig. 37. For the depth-first tree, the maximal degree is fairly low (≤ 4) and remains approximately constant for all transmission ranges. For the

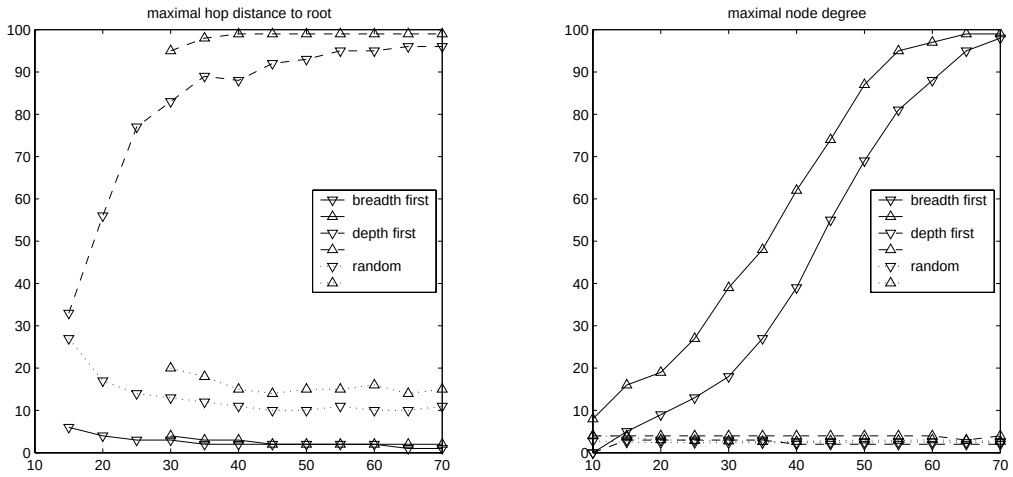


Fig. 37: Maximal distance to the reference node and maximal number of communication events per node and per real-time interval Δt. For every transmission range, 50 random networks are generated, the figure shows maximal ($\triangle$) and minimal ($\bigtriangledown$) values.

breadth-first tree, the maximal degree is small for a small transmission range and then quickly increases. At the transmission range of $50\sqrt{2} \approx 71$, the reference node can reach every other node in the area and its degree is maximal, while all other nodes have degree 1.

Now consider the maximal hop distance to the reference node. At a transmission range of 10, there are nodes that cannot communicate with any other node, thus the maximal distance is not shown. In some of the 50 graphs generated for every transmission range, this occurs up to a transmission range of 30. 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 hop distance is much larger and increases quickly (≥ 83 for a transmission range of at least 30).

The interval-based approach distributes the communication events among the nodes equally well as the depth-first-tree approach, which is far better than in the case of the breadth-first-tree. The maximal hop distance, and thus the worst-case accuracy, of the interval-based approach 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).

4.2.4 Summary

In this section, we have presented interval-based synchronization as an approach for solving the multihop synchronization problem as it has been defined in Sect. 4.1. It has been shown that the simple algorithm $\mathcal{A}^{\mathrm{IM}}$, which had been presented in [MO83], solves the problem worst-case-optimally. We have also analyzed various tree-based synchronization approaches and derived lower bounds for the worst-case accuracy they can achieve. Finally, we have

compared interval-based and tree-based algorithms and identified various advantages of the interval-based approach:

- The interval-based algorithm $\mathcal{A}^{\mathrm{IM}}$ is worst-case-optimal in arbitrary scenarios and thus allows to make optimal use of every scenario. An arbitrary scenario can result from using a traffic pattern for synchronization that is generated by some other, non-synchronization functionality of the network. It has been shown that tree-based algorithms cannot be optimal in arbitrary scenarios.

- For a specified frequency of communication events, it has been shown (see Fig. 37) that arbitrary communication events provide better synchronization accuracy than a depth-first tree. The accuracy is only slightly worse than in a breadth-first tree. The distribution of the communication events (and thus of energy consumption) is equally good as in the case of a depth-first tree and far better than in a breadth-first tree.

- Assume that some node fails due to depletion of batteries, physical damage, etc. The tree-based algorithms now have to reestablish a tree structure of the network, which is expensive in terms of communication overhead. [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. For the networks we have studied above, the number of nodes is $n = 100$ and the average number of edges, assuming a transmission range of 30, is approximately 1100. Thus, the communication overhead of breadth-first tree construction is approximately 13270 communication events and that of the depth-first tree is approximately 4400 communication events. In contrast, *the interval-based approach has zero overhead.*

4.3 Back-Path Interval Synchronization

In this section, we present and analyze an improved version of the algorithm $\mathcal{I}^{\mathrm{IM}}$, the Back-Path Interval-Synchronization Algorithm $\mathcal{I}^{\mathrm{BP}}$. Algorithm $\mathcal{I}^{\mathrm{BP}}$ is worst-case-optimal like algorithm $\mathcal{I}^{\mathrm{IM}}$, but achieves better results in typical cases.

4.3.1 Worst Case and Typical Cases

Theorems 17 and 19 state bounds for the worst-case accuracy in an arbitrary scenario. The different "cases" are different traces that correspond to a particular scenario and these traces differ in the drift rates of the nodes. In the proof of Thm. 19, it has been shown that the worst-case trace corresponding to a particular scenario is the trace in which all nodes have the same drift rate and in which this drift rate is maximal. Why is that so?

Assume that two nodes N_u and N_v communicate twice without communicating with any other nodes in between. Such a scenario is shown in of Fig. 38(a). Both nodes execute algorithm $\mathcal{I}^{\mathrm{IM}}$. Immediately after the first communication event (at t_1), the lower- and upper-bound clocks of both nodes have the same values. Let $\check{E}^u = \check{C}^u(H^u(t)) - t$ be the error of the lower-bound clock $\check{C}^u$ of node N_u, and let $\hat{E}^u = \hat{C}^u(H^u(t)) - t$ be the error of the corresponding upper-bound clock. Similarly for node N_v.

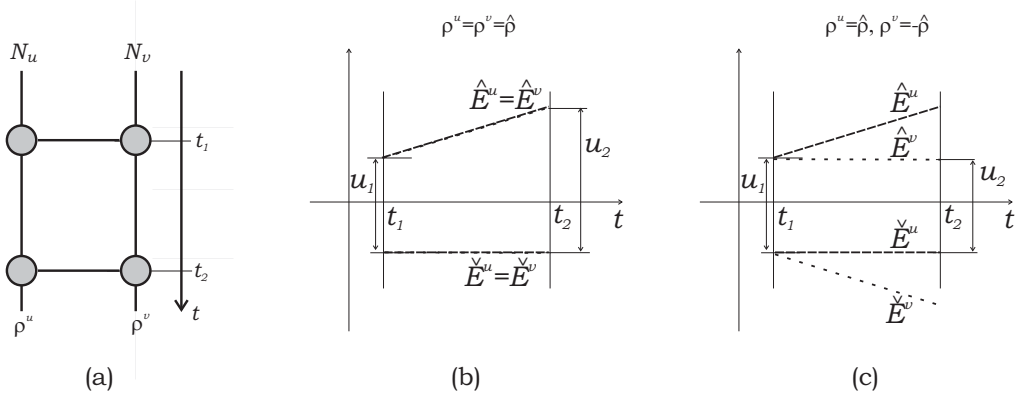


Fig. 38: (a) Scenario with two nodes. (b,c) Error of the lower- and upper-bound clocks of nodes N_u (dashed) and N_v (dotted), computed by $\mathcal{I}^{\mathrm{IM}}$ with $\rho^u = \hat{\rho}$ and different ρ^v.

In Fig. 38(b) and (c), the errors of the lower- and upper-bound clocks of the two nodes are shown for two different cases, each with a different value of the drift rate ρ^v. If $\rho^v = \rho^u$, then the second communication event does not have any effect on either node's clocks, since both the lower- and the upper-bound clocks show exactly the same values. This is the case we have referred to as the worst case. If the drift rate ρ^v is different from ρ^u, then the nodes can improve the uncertainty at the second communication event. In the extreme case of $\rho^u = \hat{\rho}$ and $\rho^v = -\hat{\rho}$, the uncertainty after the second communication event is exactly the same as after the first communication event; in this case, the two nodes have maintained the uncertainty over time without any loss. This is the best case.

If we look at this best case a little bit closer, we see that on a node with minimal drift, the error of the upper-bound clock does not increase (see Eq. 4.2). On a node with maximal drift, the error of the lower-bound clock does not increase (see Eq. 4.3). Thus *diversity* in the drift rate of the local clocks in a network minimizes the uncertainty achieved by algorithm $\mathcal{I}^{\mathrm{IM}}$, and thus also improves the accuracy achieved by algorithm $\mathcal{A}^{\mathrm{IM}}$.

4.3.2 Computing Back-Paths

In Sect. 4.3.1, we have shown that algorithm $\mathcal{I}^{\mathrm{IM}}$ achieves a better uncertainty in typical cases than in the worst case because it makes use of the rate diversity among the nodes in a network. In this section, we present algorithm $\mathcal{I}^{\mathrm{BP}}$, which is a modification of algorithm $\mathcal{I}^{\mathrm{IM}}$ and which makes even better use of this

diversity.

Def. 35: **(Algorithm $\mathcal{I}^{\mathrm{BP}}$)** *In a network of nodes executing the interval-based algorithm $\mathcal{I}^{\mathrm{BP}}$, a communication event between nodes N_u and N_v consists of a sequence of two message exchanges as specified in Alg. 7. The reference node does not use Alg. 7; instead, it always sends $(0, \infty)$ in the first message exchange, and the current real time (t, t) in the second.*

In the following, we first describe the basic idea of algorithm $\mathcal{I}^{\mathrm{BP}}$ and then give a concrete example to illustrate the various steps of Alg. 7.

Principle of Algorithm $\mathcal{I}^{\mathrm{BP}}$

In contrast to algorithm $\mathcal{I}^{\mathrm{IM}}$, algorithm $\mathcal{I}^{\mathrm{BP}}$ stores information about communication events in its memory. A node N_u stores for every other node N_v in the network the local time and the lower- and upper-bound times of the last encounter.

Whenever N_u communicates with some other node, it not only tries to improve its current lower- and upper-bounds, but also the bounds of all previous communication events stored in memory.

When node N_u meets for the second time with some node N_v, the nodes first exchange the bounds corresponding to the previous communication event between N_u and N_v, which they have stored locally. Due to some other communication event with a third node, these bounds may have improved since the nodes have met the last time. If so, these improved bounds are now used to try to improve also the bounds of all other communication events stored in memory and also the current bounds.

Only now the nodes N_u and N_v exchange and intersect their current bounds. And again, the nodes try to improve the bounds of all communication events stored in memory.

To further explain how algorithm $\mathcal{I}^{\mathrm{BP}}$ works, we now give a concrete example. The example will also explain why the algorithm is dubbed "back-path" interval-synchronization algorithm.

Ex. 13: *Assume a scenario as it is shown in Fig. 39 (a). This scenario is the same as that in Fig. 38, except that node N_u has an additional communication event with the reference node between the two communication events with node N_v, and thus the uncertainty of node N_u immediately after t_2 is zero. Let the drift rates $\rho^u = \hat{\rho}$ and $\rho^v = -\hat{\rho}$.*

Fig. 39(b) shows the lower- and upper-bound clocks and the uncertainties if nodes N_u and N_v use algorithm $\mathcal{I}^{\mathrm{IM}}$. As node N_u has maximal drift, its lower-bound clock can maintain zero error between t_2 and t_3, while the error of its upper-bound clock increases. The error of node N_v's clocks is not affected by the communication event of node N_u with the reference node.

We now explain step by step how algorithm $\mathcal{I}^{\mathrm{BP}}$ achieves a smaller uncertainty after t_3, which is illustrated in Fig. 39(c).

Algorithm 7 Algorithm $\mathcal{I}^{\mathrm{BP}}$

Parameters: Number of nodes in the network n,
bound on clock drift $\hat{\rho}$
State: Local times of prev. comm. events $H[0 \ldots n] = 0$,
lower-bound times of prev. comm. events $L[0 \ldots n] = 0$,
upper-bound times of prev. comm. events $U[0 \ldots n] = \infty$
identifier of last node encountered l
Input: Current lower- and upper-bound clock times $(\check{c}_i^v, \hat{c}_i^v)$,
previous lower- and upper-bound clock times $(\check{c}_j^v, \hat{c}_j^v)$
Output: Lower-bound clock $\check{C}_i^u$,
Upper-bound clock $\hat{C}_i^u$

// STEP 1: Exchange previous bounds
send to N_v bounds $(L[v], U[v])$, **receive** from N_v bounds $(\check{c}_j^v, \hat{c}_j^v)$

// Update all lower- and upper-bound times using $(\check{c}_j^v, \hat{c}_j^v)$
for $w := 1$ **to** n **do**
  if $H[w] < H[v]$ **then**
    $(L[w], U[w]) := \left(\max\left(L[w], \check{c}_j^v + \frac{H[w]-H[v]}{1-\hat{\rho}}\right), \min\left(U[w], \hat{c}_j^v + \frac{H[w]-H[v]}{1+\hat{\rho}}\right)\right)$
  else
    $(L[w], U[w]) := \left(\max\left(L[w], \check{c}_j^v + \frac{H[w]-H[v]}{1+\hat{\rho}}\right), \min\left(U[w], \hat{c}_j^v + \frac{H[w]-H[v]}{1-\hat{\rho}}\right)\right)$
  end if
end for

// compute current bounds
if l not initialized **then**
  $(\check{c}_i^u, \hat{c}_i^u) := (0, \infty)$
else
  $(\check{c}_i^u, \hat{c}_i^u) := \left(L[l] + \frac{h_i^u - H[l]}{1+\hat{\rho}}, U[l] + \frac{h_i^u - H[l]}{1-\hat{\rho}}\right)$
end if

// STEP 2: Exchange current bounds
send to N_v bounds $(\check{c}_i^u, \hat{c}_i^u)$, **receive** from N_v bounds $(\check{c}_i^v, \hat{c}_i^v)$

// intersect current bounds
$(L[v], U[v], H[v]) := (\max(\check{c}_i^u, \check{c}_i^v), \min(\hat{c}_i^u, \hat{c}_i^v), h_i^u)$
$l := v$

// Update all lower- and upper-bound times using $(L[v], U[v])$
for $w := 1$ **to** n **do**
  $(L[w], U[w]) := \left(\max(L[w], L[v] + \frac{H[w]-H[v]}{1-\hat{\rho}}), \min(U[w], U[v] + \frac{H[w]-H[v]}{1+\hat{\rho}})\right)$
end for

// Finally compute the clocks
$\left(\check{C}_i^u(H^u(t)), \hat{C}_i^u(H^u(t))\right) := \left(L[v] + \frac{H^u(t)-H[v]}{1+\hat{\rho}}, U[v] + \frac{H^u(t)-H[v]}{1-\hat{\rho}}\right)$

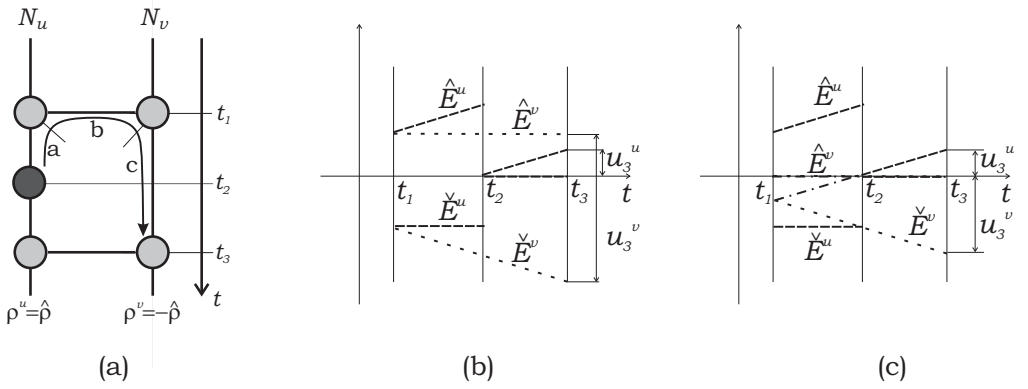


Fig. 39: *Improvements by computing back paths.* (a) Scenario. (b) Errors and uncertainties achieved by $\mathcal{I}^{\mathrm{IM}}$. (c) Errors and uncertainties achieved by $\mathcal{I}^{\mathrm{BP}}$.

After t_1, node N_u has stored lower- and upper-bound of the first communication event with N_v in $L[v]$ and $U[v]$. At t_2, node N_u improves its current bounds and stores them in memory $L[0] = U[0] = t_2$. It then also re-evaluates the bounds for the past event at t_1 (second **for** *loop in Alg. 7), that is the lower-bound time is re-evaluated as $L[v] = \max(L[v], t_2 + (h_1^u - h_2^u)/(1 - \hat{\rho}))$, and the upper-bound time as $U[v] = \min(U[v], t_2 + (h_1^u - h_2^u)/(1 + \hat{\rho}))$. Note that $h_1^u - h_2^u$ is negative. Consequently, the rate-correction factors for the lower and the upper bound are exchanged: While $t_2 \geq t_1 + (h_2^u - h_1^u)/(1 + \hat{\rho})$, it is $t_1 \geq t_2 + (h_1^u - h_2^u)/(1 - \hat{\rho})$ and while $t_2 \leq t_1 + (h_2^u - h_1^u)/(1 - \hat{\rho})$, it is $t_1 \leq t_2 + (h_1^u - h_2^u)/(1 + \hat{\rho})$. This means that a node with maximal drift can maintain a lower bound without loss when time increases and an upper bound when time decreases. For a node with minimal drift, the opposite is true. Thus, node N_u computes $U[v] = t_1$ with zero error. This is illustrated by the dash-dotted line in Fig. 39(c).*

At t_3, node N_v first receives the improved bounds for the communication event at t_1. In particular, it receives the perfectly accurate upper bound t_1. It then reevaluates its current bounds, and as it has minimal drift, it computes the current upper bound $\hat{c}_3^v = t_1 + (h_3^v - h_1^v)/(1 - \hat{\rho}) = t_3$ with zero error. On the other hand, node N_u running at maximal speed computes the lower bound $\check{c}_3^u = t_2 + (h_3^u - h_2^u)/(1 + \hat{\rho}) = t_3$, also with zero error. When the nodes finally intersect their current bounds, they arrive at t_3 for both the lower and the upper bound, thus achieve zero uncertainty.

While also algorithm $\mathcal{I}^{\mathrm{IM}}$ computes an accurate lower bound at t_3, the upper bound is different. Algorithm $\mathcal{I}^{\mathrm{BP}}$ is better, because it computed the upper bound over the "back path" shown in Fig. 39(a). At t_2, it computed the first section (a) of this path, and at t_3, the second section (b) and the last section (c).

4.3.3 Analysis

We show that in every scenario, algorithm $\mathcal{I}^{\mathrm{BP}}$ performs at least as well as the algorithm $\mathcal{I}^{\mathrm{IM}}$. Therefore the estimate-based algorithm $\mathcal{A}^{\mathrm{BP}}$, derived according to Def. 33, is worst-case-optimal. By means of a simple scenario, we illustrate how and when $\mathcal{I}^{\mathrm{BP}}$ can provide better synchronization than $\mathcal{I}^{\mathrm{IM}}$.

Thm. 21: (Worst-Case-Optimality of $\mathcal{A}^{\mathrm{BP}}$) *The uncertainty achieved by algorithm $\mathcal{I}^{\mathrm{BP}}$ is never worse than the uncertainty achieved by algorithm $\mathcal{I}^{\mathrm{IM}}$. Thus algorithm $\mathcal{A}^{\mathrm{BP}}$ is worst-case-optimal.*

Proof: We show that the lower-bound time computed by $\mathcal{I}^{\mathrm{BP}}$ is always equal or higher than that computed by $\mathcal{I}^{\mathrm{IM}}$. Analogously, it can be shown that the upper-bound time is always equal or smaller. Then the accuracy of $\mathcal{A}^{\mathrm{BP}}$ is never worse than that of $\mathcal{A}^{\mathrm{IM}}$. Since $\mathcal{A}^{\mathrm{IM}}$ is worst-case-optimal (by Thm. 20), also $\mathcal{A}^{\mathrm{BP}}$ is worst-case-optimal.

To show that the lower-bound time computed by $\mathcal{I}^{\mathrm{BP}}$ is always equal or higher than that computed by $\mathcal{I}^{\mathrm{IM}}$, assume that Alg. 7 consists only of the lines labeled with "intersect current bounds" and "finally compute the clocks". Then it is exactly equal to algorithm $\mathcal{I}^{\mathrm{IM}}$ according to Def. 34. When adding the other parts of Alg. 7 again, we see that these instructions either increase the lower bound or do nothing. Therefore the final lower bound must be equal or larger than that computed by $\mathcal{I}^{\mathrm{IM}}$. □

Improvement in Typical Cases

The improvement in terms of uncertainty of $\mathcal{I}^{\mathrm{BP}}$ over $\mathcal{I}^{\mathrm{IM}}$ in Ex. 13 is 100%. The reason for this impressive improvement is that the communication event at t_2 is with the reference node and that the rate difference between nodes N_u and N_v is maximal. In the following, we systematically study the improvement in the same kind of scenario, but assuming somewhat less favorable conditions.

In Fig. 40, a parameterized scenario similar to that of Ex. 13 is shown. The parameter x specifies at what time node N_u communicates with a third node. Assume that after the first communication event between N_u and N_v, the uncertainty is u and after the communication event of N_u with a third node it is $y \cdot u$. Fig. 40 illustrates the improvement of the uncertainty computed by $\mathcal{I}^{\mathrm{BP}}$ relative to that computed by $\mathcal{I}^{\mathrm{IM}}$ in function of the parameters x and y and the drift rates ρ^u and ρ^v.

In the upper-right corner of Fig. 40, the communication event of N_u with a third node occurs immediately after the communication event between N_u and N_v ($x \approx 0$), and this third node is the reference node, thus provides zero uncertainty ($y = 0$). In this case, the improvement of $\mathcal{I}^{\mathrm{BP}}$ is 100% if the nodes' rate-difference is maximal. The improvement is 0% if the drifts are equal.

The graphs in the center row of Fig. 40 explore the effect of the position of the communication event of node N_u with a third node. The average improvement decreases with increasing distance between this event and the first communication event between N_u and N_v. The maximal improvement is still

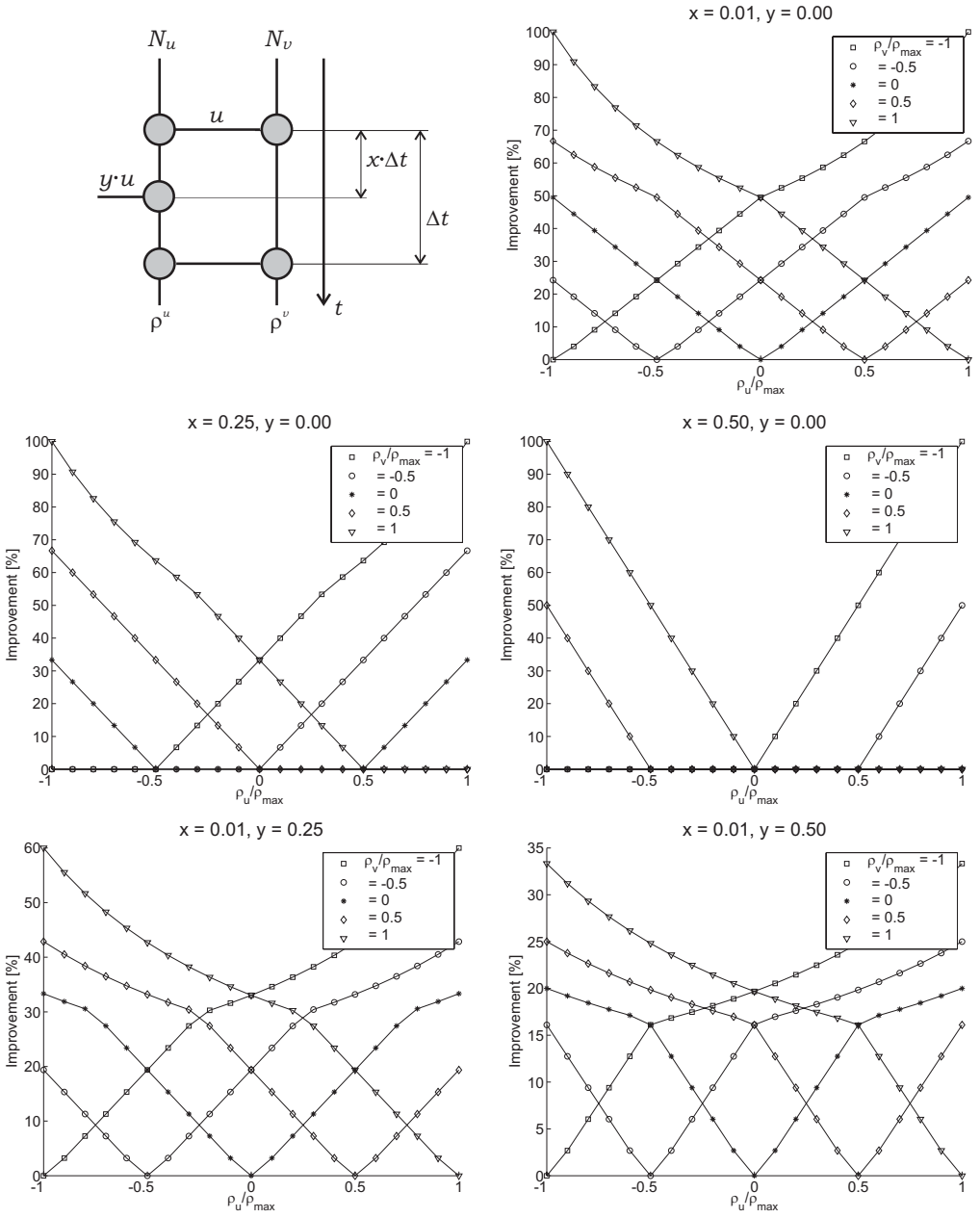


Fig. 40: *Improvement of $\mathcal{I}^{\mathrm{BP}}$ in the simple scenario depicted in the upper left corner.* In all figures, the drift rates of both nodes are varied between $-\hat{\rho}$ and $\hat{\rho}$. Each graph shows one combination of the parameters x and y.

100% for maximal drift-rate difference, but the average (over all pairs of ρ^u and ρ^v) decreases.

The graphs in the bottom row of Fig. 40 explore the effect of the time uncertainty achieved at the communication event with the third node. With increasing uncertainty, the maximal improvement decreases, for example at $y = 0.5$, $\mathcal{I}^{\mathrm{BP}}$ can only achieve 34% less uncertainty than $\mathcal{I}^{\mathrm{IM}}$ if the drift-rate difference is

maximal.

4.3.4 Experimental Study: Comparison of $\mathcal{I}^{\mathrm{BP}}$ and $\mathcal{I}^{\mathrm{IM}}$

We now compare the interval-based algorithms $\mathcal{I}^{\mathrm{BP}}$ and $\mathcal{I}^{\mathrm{IM}}$ in large networks. The networks are generated in the same way as in Sect. 4.2.3, where the *worst-case accuracy* achieved by interval-based CSAs is compared to that achieved by tree-based CSAs. In contrast, we now compare the accuracy in *typical cases*, that is in networks of nodes that have different, non-maximal drift rates.

Experimental Setup

Traces of networks with 100 nodes and with a length of 500 hours are generated. The nodes are placed randomly with a uniform distribution in a square area of width 100. Every node has a constant drift rate which is chosen randomly with a uniform distribution from the range $[-\hat{\rho}, \hat{\rho}]$ with $\hat{\rho} = 100\mathrm{ppm}$. In these scenarios, the algorithms $\mathcal{I}^{\mathrm{BP}}$ and $\mathcal{I}^{\mathrm{IM}}$ are simulated and the average uncertainty of the nodes is evaluated. By *average* we here understand the following: The uncertainty is evaluated immediately after every communication event in the network. If the uncertainty of a particular node after a particular communication event is infinite, it is not considered in the calculation of the average. All other values have equal weight concerning the average calculation.

In this evaluation, the following parameters are varied: (i) The transmission range, influencing the number of neighbors a particular node can communicate with (see Fig. 36). (ii) The frequency of communication events f_C per node. (iii) The number of reference nodes and (iv) the frequency of communication events f_R for reference nodes. For every parameter set, 50 traces are generated and the average (of the average) uncertainty is evaluated.

The last two parameters, i.e., the number of reference nodes and their frequency of communication events, require some explanation. In Sect. 4.2.3, only one reference node has been used, since tree-based CSAs require that there is a unique root of the tree. In contrast, interval-based algorithms do not make any constraints on the number of reference nodes.

Results

Figure 41 displays the average uncertainty achieved by the two algorithms with a transmission range of 20. The uncertainty decreases when communication events becomes more frequent. Increasing the communication frequency among client nodes decreases the uncertainty to a constant level, while increasing the reference-communication frequency f_R leads to arbitrarily small uncertainties. Below, the relative improvement of $\mathcal{I}^{\mathrm{BP}}$ over $\mathcal{I}^{\mathrm{IM}}$ is displayed. The improvement increases when the client nodes communicate more frequently among each other. The improvement decreases if the client nodes communicate often with the reference nodes. The improvement also decreases with increasing number of reference nodes.

Figure 42 displays the relative improvement of $\mathcal{I}^{\mathrm{BP}}$ over $\mathcal{I}^{\mathrm{IM}}$ as a function of

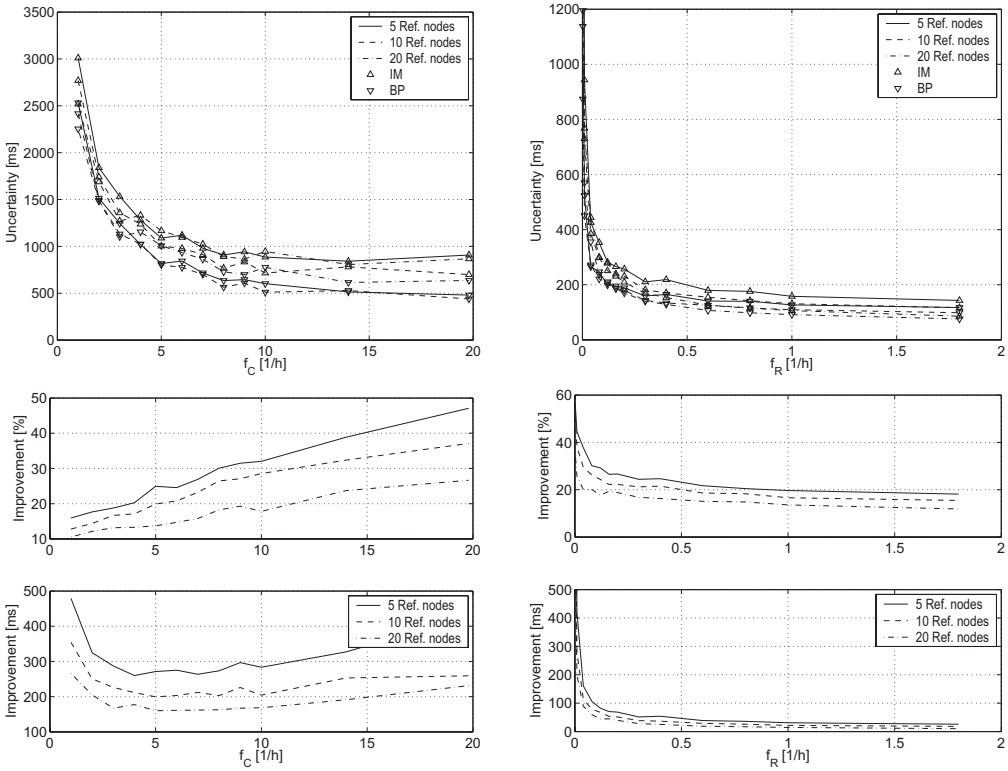


Fig. 41: *Time uncertainty and relative improvement of $\mathcal{I}^{\mathrm{BP}}$ over $\mathcal{I}^{\mathrm{IM}}$.* Left: As a function of the frequency of communication among client nodes f_C , with $f_R = 1/h$. Right: As as function of the frequency of communication with reference nodes f_R, with $f_C = 10/h$.

the transmission range. The improvement increases significantly up to a transmission range of ≈ 15 and remains approximately constant at larger values.

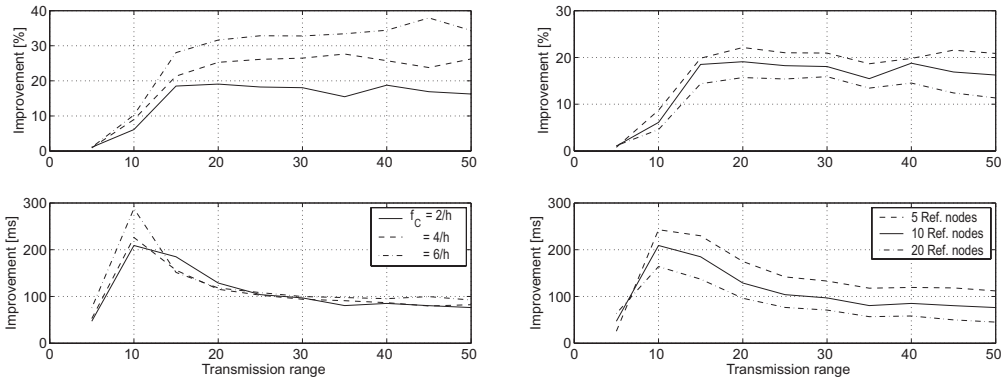


Fig. 42: *Improvement as a function of the transmission ranges.* Left: With different communication frequencies f_C, 10 reference nodes and $f_R = 1/h$. Right: With different numbers of reference nodes, $f_C = 4/h$ and $f_R = 1/h$.

Discussion

Depending on the parameters of the traces, the improvement of algorithm $\mathcal{I}^{\mathrm{BP}}$ over $\mathcal{I}^{\mathrm{IM}}$ can be substantial, that is the average uncertainty can be more than 50% better. For other parameters, the improvement is only marginal, that is smaller than 10%. As expected (see Thm. 21), the improvement is always positive, i.e., $\mathcal{I}^{\mathrm{BP}}$ always achieves a smaller uncertainty than $\mathcal{I}^{\mathrm{IM}}$. In the following, we discuss how the improvement depends on the parameters of the traces.

Consider the left column of Fig. 41. When the frequency of communication events increases, the improvement increases, too. As the nodes communicate with a randomly chosen neighbor, the number of neighbors with which a particular node communicates also increases when the frequency of communication increases. In Sect. 4.3.3, we have identified the diversity of the drift rates among the nodes that communicate as a source of improvements in the uncertainty. We conjecture that this increased diversity of drift rates is responsible for the increasing improvement.

On the other hand, the increasing frequency of communication events with reference nodes *decreases* the improvement in the uncertainty, as can be seen in the right column of Fig. 41. As we evaluate the uncertainty as the average of the uncertainties immediately after all communication events, the influence of the communication events with a reference node increases with the frequency of such events. The uncertainty immediately after such an event is zero, both for $\mathcal{I}^{\mathrm{BP}}$ and for $\mathcal{I}^{\mathrm{IM}}$. We conjecture that this is the reason why the improvement decreases. The same effect presumably is also responsible for the fact that in all diagrams of Figs. 41 and 42, the improvement is smaller for a large number of reference nodes than for a smaller number of reference nodes.

As shown in Fig. 42, the improvement first increases with the transmission rate and then remains approximately constant. There are at least two plausible explanations for the small improvement at small transmission ranges: (i) Many nodes are not connected with a reference node, neither directly nor via other intermediate client nodes (see Fig. 36, top row). These nodes are not included in the evaluation of the average uncertainty, since their uncertainty is infinite. The nodes that are connected with a reference node are rather close to the reference node in terms of hop distance. These nodes thus have a small uncertainty for both $\mathcal{I}^{\mathrm{BP}}$ and for $\mathcal{I}^{\mathrm{IM}}$, thus the improvement is small. (ii) When the transmission rate is small, all nodes have a small number of neighbors. Thus, also the drift diversity is small and therefore also the improvement of $\mathcal{I}^{\mathrm{BP}}$.

The experiment has shown that the improvement of algorithm $\mathcal{I}^{\mathrm{BP}}$ over algorithm $\mathcal{I}^{\mathrm{IM}}$ is large when the drift diversity among neighboring client nodes is large and the hop distance to a reference node is relatively large. We thus expect that in scenarios with mobility of client nodes, the improvement of $\mathcal{I}^{\mathrm{BP}}$ over $\mathcal{I}^{\mathrm{IM}}$ is even more important than in the static scenarios we have studied here.

4.4 Summary

In this chapter, we have introduced a system model for multihop synchronization problems. The model makes a high-level abstraction of the communication: Two nodes can exchange an arbitrary amount of information with zero delay. This abstraction has not been used in previous work on clock synchronization. While it is somewhat unrealistic, this model has allowed to derive a lower bound on the achievable accuracy in arbitrary networks and arbitrary scenarios. We have specialized this lower bound for networks with a chain or tree topology.

We have presented interval-based synchronization as a novel, completely local approach to the multihop synchronization problem. In contrast to tree-based and clustering synchronization schemes, interval-based synchronization is free of network-wide configuration. It is thus inherently tolerant against node or link failures and naturally copes with node mobility. It has been shown that worst-case-optimal interval-based synchronization can be realized at very low cost, using algorithm $\mathcal{I}^{\mathrm{IM}}$.

Finally, we have presented the more complicated interval-based algorithm $\mathcal{I}^{\mathrm{BP}}$, which performs at least equally well as $\mathcal{I}^{\mathrm{IM}}$ in all scenarios. In typical scenarios, it can achieve substantially better performance. The cost of this improvement is a memory and computational overhead that is proportional to the number of nodes in the network.

5

Synthesis

In this chapter, it is shown how the results of the last two chapters can be combined to construct a *complete and practical time service*. In Chap. 3, we have presented algorithms that deal with variable message delay and clock drift in a simple single-hop scenario. In Chap. 4, we have additionally dealt with the problem of organizing synchronization in large multihop networks, but we have ignored message delays. A complete time service has to deal with all three problems, that is with variable delay, clock drift, and with the organization of the synchronization process. A complete time service can be constructed if we manage to combine the Local Selection CSAs from Chap. 3 with the interval-based CSAs from Chap. 4.

The main difference between the system models used in the previous two chapters is that in Chap. 3, the emphasis is put on various models for the delay of time-stamp messages, whereas in Chap. 4, message delays are all zero. To reconcile these two viewpoints, we assume now that these two models are based on different time scales, as illustrated in Fig. 43: While the communication events introduced in Chap. 4 occur in intervals of seconds, minutes or hours, the time-stamp messages introduced in Chap. 3 are sent in intervals of milliseconds. A communication event can thus be interpreted as a high-level abstraction of a sequence of time-stamp message exchanges between two nodes. In the following, it is shown how a communication event can be constructed from two sequences of time-stamp messages using the Local-Selection CSAs. Based on these constructed communication events, the interval-based CSAs can then be used exactly as described in the last chapter.

The problem of constructing communication events is split in two parts: In Sect. 5.1, it is shown how lower and upper bounds on real time have to be transformed such that they are valid at the reception of a message with non-zero delay. The proposed solution makes the assumption that the receiving node

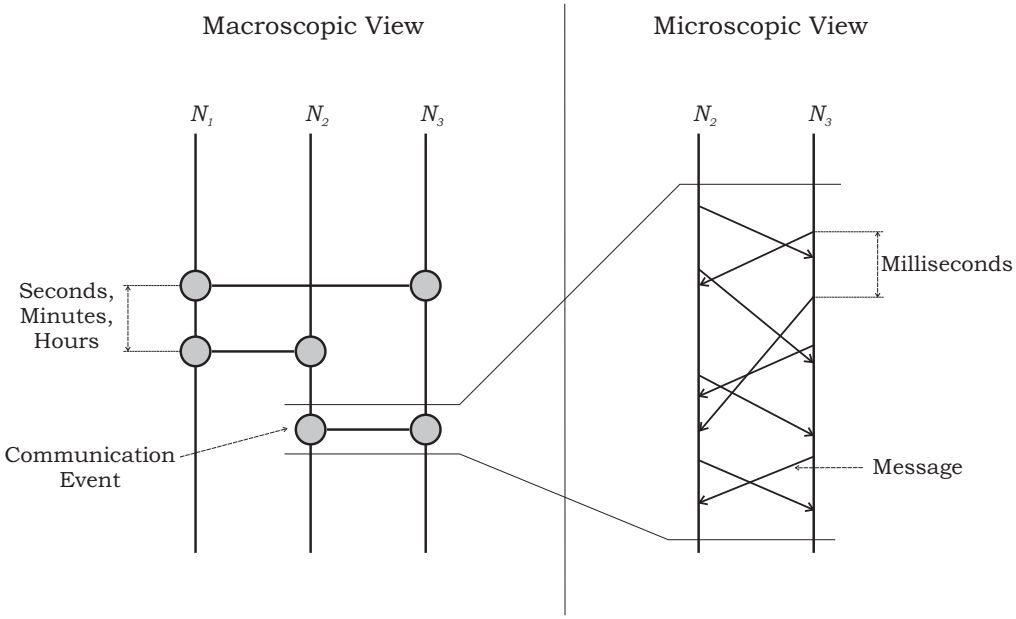


Fig. 43: Combining multihop synchronization (macroscopic view) with point-to-point synchronization (microscopic view). A communication event in the macroscopic view consists of a bidirectional sequence of message exchanges in the microscopic view.

maintains lower- and upper-bound clocks on the sending node's local clock. In Sect. 5.2, it is shown how the Local Selection algorithms from Chap. 3 can be adapted for maintaining such lower- and upper-bound clocks on the sending node's local clock.

5.1 Multihop Synchronization with Message Delay

In this section, it is shown how valid lower and upper bounds $\check{c}_s^v$ and $\hat{c}_s^v$ on the real time t_s when node N_v sends a message have to be transformed by node N_u, such that they are valid at the real time t_r when the message is received. This is not a trivial task, since an upper bound on real time that is valid at the time of sending a message is not necessarily valid at the time of receiving this message.

Thm. 22: **(Bounds Transformation)** *Let a node N_v send a message containing the triple $(h_s^v, \check{c}_s^v, \hat{c}_s^v)$ to N_u, where h_s^v is the local time of N_v at real time t_s, that is the time of sending this message. Let $\check{c}_s^v$ and $\hat{c}_s^v$ be the lower and upper bounds of node N_v on real time t_s . Let N_u's logical clocks $\check{C}^{u,v}$ and $\hat{C}^{u,v}$ be valid lower- and upper-bound clocks on N_v's local clock H^v, i.e., $\check{C}^{u,v}(H^u(t)) \leq H^v(t) \leq \hat{C}^{u,v}(H^u(t))$ for all t. Then the node N_u computes valid lower and upper bounds $\check{c}_r^u$ and $\hat{c}_r^u$ on real time t_r using*

$$\check{c}_r^u = \check{c}_s^v + \frac{\check{c}_r^{u,v} - h_s^v}{1+\hat{\rho}} \tag{5.1}$$

$$\hat{c}_r^u = \hat{c}_s^v + \frac{\hat{c}_r^{u,v} - h_s^v}{1-\hat{\rho}} \ . \tag{5.2}$$

The uncertainty u_r^u at the time of receiving the message at N_u is

$$u_r^u \leq u_s^v + u_r^{u,v} + 2\hat{\rho}(\hat{c}_r^{u,v} - h_s^v) \ , \tag{5.3}$$

where $u^{u,v}$ is the uncertainty of the logical clocks $\check{C}^{u,v}$ and $\hat{C}^{u,v}$, and u_s^v is the uncertainty of the bounds on real time of node N_v at the time of sending the message.

Proof: We give the proof for Eq. 5.1, the proof for Eq. 5.2 can be derived analogously.

It is assumed that $\check{C}^{u,v}$ is a valid lower-bound clock on the local clock H^v of node N_v. Thus the local time of N_v at the time of the message reception is $h_r^v \geq \check{c}_r^{u,v}$. As the local time at the time of sending the message is h_s^v, the local-time difference corresponding to the delay of the message is $h_r^v - h_s^v \geq \check{c}_r^{u,v} - h_s^v$, and the corresponding real time difference is $t_r - t_s \geq \frac{\check{c}_r^{u,v} - h_s^v}{1+\hat{\rho}}$. Therefore if $t_s \geq \check{c}_s^v$, then also $t_r \geq \check{c}_s^v + \frac{\check{c}_r^{u,v} - h_s^v}{1+\hat{\rho}} = \check{c}_r^u$.

To show that Eq. 5.3 is valid, we compute the difference between Eq. 5.2 and Eq. 5.1, which results in $u_r^u = \hat{c}_r^u - \check{c}_r^u = \hat{c}_s^v - \check{c}_s^v + \frac{\hat{c}_r^{u,v}}{1-\hat{\rho}} - \frac{\check{c}_r^{u,v}}{1+\hat{\rho}} - \check{c}_s^v(\frac{1}{1-\hat{\rho}} - \frac{1}{1+\hat{\rho}})$. Thus $u_r^u = u_s^v + \frac{1}{1-\hat{\rho}^2}\left(u_r^{u,v} + \hat{\rho}(\hat{c}_r^{u,v} - h_s^v + \check{c}_r^{u,v} - h_s^v)\right)$. Since the maximal drift rate $\hat{\rho}$ is very small, we approximate $\frac{1}{1-\hat{\rho}^2} \approx 1$. Replacing $\check{c}_s^{u,v}$ with $\hat{c}_s^{u,v}$, we get the inequality $u_r^u \leq u_s^v + u_r^{u,v} + 2\hat{\rho}(\hat{c}_r^{u,v} - h_s^v)$. □

Discussion

We have shown that a communication event can be constructed from a message with delay containing the triple $(h_s^v, \check{c}_s^v, \hat{c}_s^v)$, if the receiving node N_u has the logical clocks $\check{C}^{u,v}$ and $\hat{C}^{u,v}$, which are lower and upper bounds on the sending node N_v's local clock H^v. In contrast to the communication event from Chap. 4, a communication event constructed using Thm. 22 does not provide both nodes involved in a communication event with exactly the same bounds.

Equation 5.3 can be interpreted as follows: The uncertainty at the message reception u_r^u is equal to the uncertainty u_s^v when the message was sent plus the uncertainty $u_r^{u,v}$ of the receiving node N_u about the sending node N_v's local time. In addition, the uncertainty has to be increases by $2\hat{\rho}(\hat{c}_r^{u,v} - h_s^v)$, which accounts for the time the message was in transit.

5.2 Local-Selection and Intervals

In this section, we discuss how the Local-Selection algorithms can be used to construct the logical clocks $\check{C}^{u,v}$ and $\hat{C}^{u,v}$ which are lower- and upper-bound

clocks on the local clock H^v of N_v.

Non-Real-Time Reference

Remember that in Chap. 3, the client node received time-stamp messages from a reference node, which has access to real time. In contrast, we now assume that a client node N_u synchronizes to the local clock of another client node N_v. We assume that N_v generates time-stamp messages containing time stamps h_s^v from its local clock H^v.

The difference between this scenario and the original setting of Chap. 3 is that the maximal drift of a client node relative to real time (as provided by a reference node) is $\hat{\rho}$, but the maximal drift of a client node relative to another client node's local clock is $2\hat{\rho}$. Based on Lemma 3, it can be shown that using any of the Local-Selection algorithms, node N_u computes a lower-bound clock $\check{C}^{u,v}$ on another node N_v's local clock if the parameter $\hat{\rho}$ used by the Local-Selection algorithms is doubled. Also the maximal drift variation $\hat{\vartheta}$ used by the Local-Selection algorithms with drift compensation has to be doubled.

Maintaining an Upper Bound

Computing an upper-bound clock $\hat{C}^{u,v}$ is more difficult. On the one hand, it is unclear how an initial upper bound $\hat{c}_r^{u,v}$ at the time of receiving a time-stamp message should be derived. On the other hand, the Local-Selection algorithms maintain a valid lower bound on the reference time, but not an upper bound. In a first step, we assume that an initial upper bound $\hat{c}_r^{u,v}$ is known and define the complementary Local-Selection algorithms, which maintain a valid upper bound on the reference time.

Def. 36: (Complementary Local-Selection Algorithm) *The complementary algorithm $\bar{\mathcal{A}}$ of a Local-Selection algorithm $\mathcal{A}$ is equal to $\mathcal{A}$, except that (i) all computed logical clocks progress faster than the reference clock, (ii) all adjustments are negative.*

Figure 44 gives an illustration of the lower- and upper-bound clocks computed by two Local-Selection algorithms and their complementaries.

To further illustrate the concept of a complementary Local-Selection algorithm, we define the complementary algorithm of the Basic Local-Selection algorithm $\mathcal{A}^{\text{ls}}$. The maximal drift rate is doubled with regard to Def. 16 in order to allow synchronization to another node's local clock.

Ex. 14: (Complementary Basic Local Selection) *The complementary $\bar{\mathcal{A}}^{\text{ls}}$ of the Basic Local-Selection algorithm $\mathcal{A}^{\text{ls}}$ is a selective estimate-based CSA (Def. 7) that computes candidate clocks C_i^* and decisions π_i from the view $\mathcal{V}_i = \{(\hat{c}_j^{u,v}, h_j^u) | j \leq i\}$ and the parameter $\hat{\rho}$ according to the following rules:*

$$C_i^*(h^u) = \hat{c}_i^{u,v} + \frac{h^u - h_i^u}{1 - 2\hat{\rho}}, \quad \forall h^u \geq h_i^u \tag{5.4}$$

$$\pi_i = (c_i^* < c_i^-), \quad \forall i > 1 \tag{5.5}$$

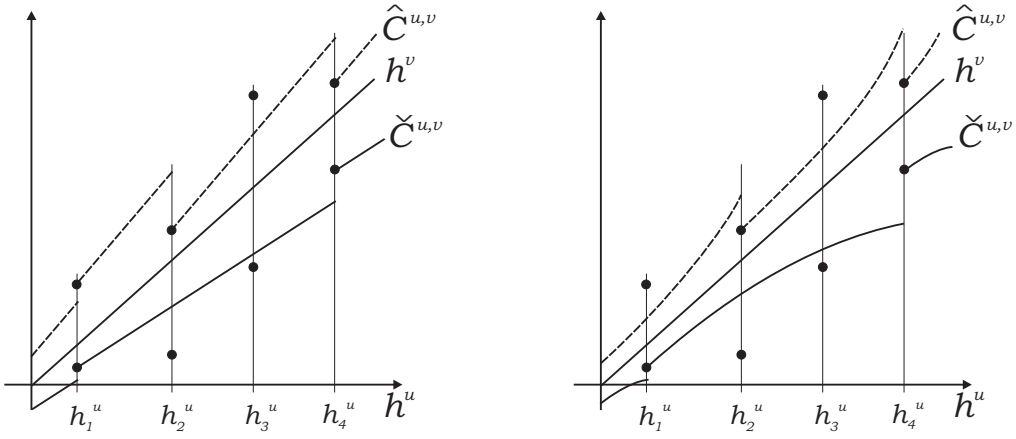


Fig. 44: Node N_u can use Local-Selection Algorithms to compute a lower-bound clock $\check{C}^{u,v}$ on the local time of node N_v. A complementary algorithm can be used to compute the upper-bound clock $\hat{C}^{u,v}$. On the left side, algorithm $\mathcal{A}^{\text{ls}}$ and its complementary $\bar{\mathcal{A}}^{\text{ls}}$ are illustrated. The right side shows the same for algorithm $\mathcal{A}^{\text{lsdc}}$.

In comparison with $\mathcal{A}^{\text{ls}}$ (see Def. 16), the decision π_i is different now: only negative adjustments are made. The candidate clocks C_i^* have the rate $\frac{1+\rho^u}{1-2\hat{\rho}}$, which is larger than the rate of H^v, that is larger than $1+\rho^v$. Thus, these clocks progress faster than the local clock H^v of node N_v.

Constructing an Initial Upper Bound

It now remains to compute the initial upper-bound $\hat{c}_i^{v,u}$. In the following, we will only consider one time-stamp message, thus we will not use the subscript i. Instead, we use the subscripts s and r to discern between the send time and the receive time of a time-stamp message (similarly as in Sect. 5.1). The basic idea is to transform a lower bound $\check{c}^{v,u}$ of node N_v on the local time of node N_u into an upper bound $\hat{c}^{u,v}$ of node N_u on the local time of node N_v. Figure 45 illustrates this procedure.

Thm. 23: *Let two nodes N_u and N_v maintain a lower-bound clock $\check{C}^{u,v}$ (respectively $\check{C}^{v,u}$) on the other node's local clock H^v (respectively H^u). Let every message from node N_v to N_u contain (i) the current local time h_s^v and (ii) the current lower-bound time on node N_u's local clock, that is $\check{c}_s^{v,u}$. At the reception of this message, node N_u computes a valid upper-bound time $\hat{c}_r^{u,v}$ according to*

$$\hat{c}_r^{u,v} = h_s^v + (h_r^u - \check{c}_s^{v,u})\frac{1+\hat{\rho}}{1-\hat{\rho}} \geq h_r^v \ . \tag{5.6}$$

Proof: Since the drifts of both nodes' local clocks are bounded, we have $(h_r^v - h_s^v) \leq (h_r^u - h_s^u)\frac{1+\hat{\rho}}{1-\hat{\rho}}$ (in Fig. 45: c is at most equal to $\frac{1+\hat{\rho}}{1-\hat{\rho}}$ times a). From the assumption that $\check{C}^{v,u}$ is a valid lower-bound clock on H^u, it follows that $\check{c}_s^{v,u} \leq h_s^u$, and thus $(h_r^u - h_s^u) \leq (h_r^u - \check{c}_s^{v,u})$ (in Fig. 45: a is at most equal to

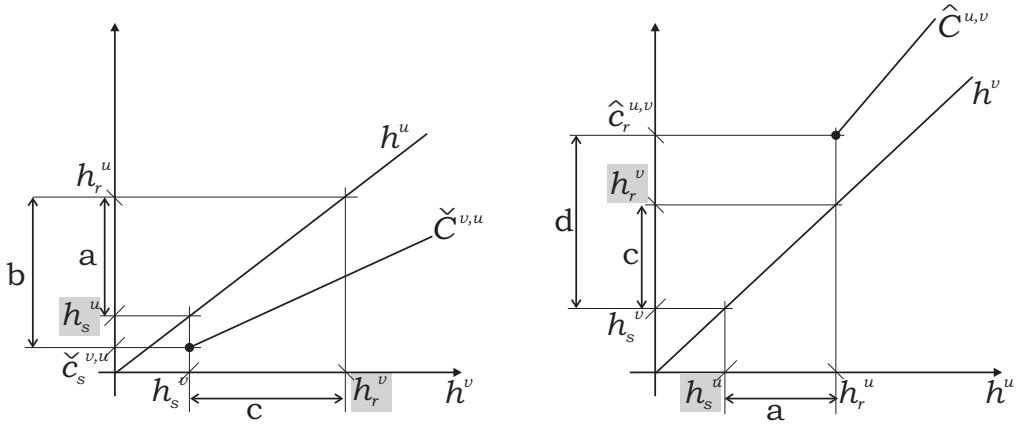


Fig. 45: Transforming a lower bound on N_u's local time into an upper bound on N_v's local time. The values with a shaded background are unknown to the nodes N_u and N_v. The left side shows the lower bound $\check{c}_s^{v,u}$ at the send time of a message, and the right side shows the corresponding upper bound $\hat{c}_r^{u,v}$ at the receive time of this message.

b). Therefore, we can write $(h_r^v - h_s^v) \leq (h_r^u - \check{c}_s^{v,u})\frac{1+\hat{\rho}}{1-\hat{\rho}}$ (in Fig. 45: c is at most equal to $\frac{1+\hat{\rho}}{1-\hat{\rho}}$ times b). This expression is equivalent to $h_r^v \leq h_s^v + (h_r^u - \check{c}_s^{v,u})\frac{1+\hat{\rho}}{1-\hat{\rho}}$ (in Fig. 45: d is exactly $\frac{1+\hat{\rho}}{1-\hat{\rho}}$ times b) . □

Discussion

In this section, we have presented two modifications to the Local-Selection algorithms. (i) By doubling the maximal drift and the maximal drift variation, Local-Selection algorithms can be used to synchronize to another client node instead of to a reference node. (ii) If two nodes symmetrically generate time-stamp messages and use Local-Selection algorithms to compute lower-bound clocks, then the complementary Local-Selection algorithms can be used to compute upper-bound clocks on the other node's local clock.

Computing an initial upper bound on another node's local clock can of course also be achieved by using a simple round-trip experiment, that is by sending a query message to a remote node and measuring the round-trip time until the reply message from the remote node is received. But the proposed solution of using two independent sequences of unidirectional time-stamp messages, transforming the lower-bound time of one node to an upper-bound time of the other node, and then employing the complementary Local-Selection algorithms has a number of advantages: (i) To achieve a small uncertainty using the round-trip experiment, it is necessary that corresponding query and reply messages both have a small delay. Our new approach achieves the same uncertainty if any one message from N_u and any one message from N_v has a small delay. It is not necessary that these messages with a short delay are immediately consecutive. (ii) Both nodes can generate messages at arbitrary times, they do not have to reply immediately to incoming queries. The time-stamp messages

can even be broadcasted to a number of neighbor nodes.

The fact that lower and upper bounds on the reference time are known has also implications on the drift-compensation mechanisms of the Local-Selection algorithms as introduced in Sect. 3.4.1. It had been shown that an upper bound on the jitter of the synchronized clock has to be known in order to compute a valid upper bound on the drift of the local clock. Such an upper bound on peak jitter can be derived from the uncertainty of the lower- and upper-bound clocks.

6

Conclusion

The goal of this thesis was to explore different approaches to derive guarantees on the quality of synchronization in a computer network. In contrast to related work, this thesis confronts empirical measurements with analytically obtained bounds. In the following, we discuss the three main contributions of this thesis.

6.1 Main Results

Algorithms for the Wireless Loudspeakers Application
In Chap. 3, the novel class of Local-Selection CSAs has been presented. The main property of these algorithms is that synchronized clocks run slower than the reference time. The algorithms require only unidirectional communication and thus an arbitrary number of client nodes can be served with a constant overhead for the reference node. The Basic Local-Selection and the Local Selection with Drift Compensation algorithms have been published in [BT02], the main property of slow synchronized clocks has been internationally patented [BD01].

In Chap. 2, we have presented a framework for the evaluation of the synchronization quality achieved by a particular clock-synchronization algorithm (CSA). The framework is novel for the evaluation of CSAs in two points: (i) It combines the advantages of measurements and simulation, that is of realistic input traces recorded in a real system and of reproducible evaluation of various CSAs using simulation on the recorded traces. (ii) The framework incorporates automatic optimization of the CSAs' parameters. It has been shown that evolutionary multiobjective optimization algorithms efficiently optimize a wide range

of CSAs without requiring adaptation to a particular CSA. Automatic parameter optimization provides to objectify the effort spent on optimizing a particular CSA and thus guarantees fair comparison of different CSAs. The framework has been published in [BT04]. An earlier version of the framework and a case study with wired Ethernet has been published in [MB03].

We have applied our novel evaluation strategy for CSAs to the application of wireless loudspeakers. We have identified a set of target performance metrics that model the stringent requirements of the application, and compared various CSAs. It has been shown that the Local-Selection algorithms can achieve the required performance robustly, that is independently of the cross-traffic in the network and with a single set of parameters for different scenarios and network-load conditions.

Improved Modeling of Message Delays

In Chap. 2, we have introduced the novel delay-interval model for the characterization of message-delay sequences. In Chap. 3, this model has been used to derive smaller, thus better, upper-bounds on peak jitter than the widely-used bounded-delay model. This is illustrated in Figure 46.

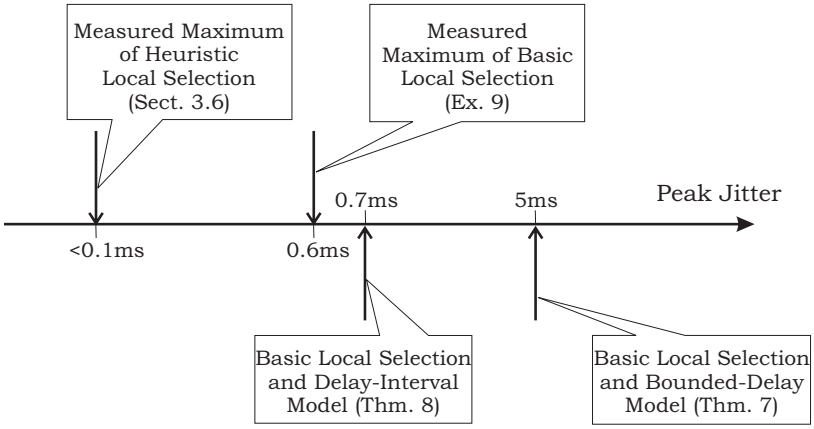


Fig. 46: A comparison of the empirically and analytically obtained "guarantees" on the peak jitter achieved by the Basic Local-Selection algorithm. The upper bound according to the novel delay-interval model describes much more accurately the actual measurement than the widely-used bounded-delay model. In the measurements, the heuristic Local-Selection algorithms achieve a far smaller peak jitter because they use drift compensation. However, no finite upper-bound can be derived because of their heuristic nature.

A Novel Approach to Multihop Synchronization

In Chapter 4, which is joint work with Lennart Meier, we have proposed a novel model for the analysis of multihop synchronization in ad-hoc networks. It has been shown that interval-based CSAs are worst-case-optimal and can achieve a similar and even better synchronization quality than the currently most popular tree-based approaches. In contrast to the tree-based approaches, the interval-

based CSAs require no system-wide configuration or collaboration. Therefore, interval-based CSAs robust against change in the system, that is node or link failures, new nodes entering the system. Also multiple reference nodes does not pose any problems. In [BMT04], we have proposed interval-based CSAs for the first time as an approach to the multihop synchronization problem, demonstrated the worst-case-optimality of Marzullo's algorithm [MO83], and introduced the novel Back-Path interval-based CSA. In [MBT04], we have identified the every-case optimal interval-based CSA, but this algorithm has not been discussed in this thesis.

In Chap. 5, we have sketched how a complete time-synchronization service can be constructed using the interval-based CSAs for system-wide synchronization based on point-to-point synchronization using the Local-Selection CSAs.

6.2 Future Perspectives

Concerning point-to-point synchronization, we have confronted empirical measurements with analytical results to discuss the suitability of message-delay and clock-drift models. Concerning the multihop synchronization problem, we have only presented analytical results and simulation, but no measurements. Measuring the synchronization quality in a large-scale wireless sensor networks poses many practical problems. Also, realistic application scenarios for such networks are not yet clearly identified and real implementations of sensor nodes are still in development. In future work, interval-based multihop synchronization schemes will have to be implemented and evaluated in realistic scenarios. Hopefully, the techniques provided in this thesis can contribute to this undertaking.

A

CSAs used for the Case Study of Sect. 3.6

Algorithm 8 Adaptive Approximate Local Selection $\mathcal{A}^{\mathrm{leak,app,adap}}$

Parameters: Initial phase ι,
leak factor λ, minimal leak factor $\lambda_{\min}$, leak adaptation factor λ_μ,
queue size q
Input: View $\mathcal{V}_i = \{(s_j, h_j) | j \in [1, i]\}$
Output: Logical clock C_i
Variables: FIFO queues Q^j, Q^h, Q^c of size q

if $i > \iota$ **then**
  $r_i := R_{i-1}(h_i)$
  if $s_i - c_i^- > 0$ **then**
    insert $(s_i - c_i^-)$ at head of Q^j
    insert s_i at head of Q^c
    insert h_i at head of Q^h
    if Q^j is full **then**
      $r_i := \frac{h_i - Q^h.tail}{s_i - Q^c.tail - \max(Q^j)} + \frac{1}{2}\hat{\vartheta}(s_i - Q^c.tail + \max(Q^j)) - 1$
      $\lambda := (1 - \lambda_\mu)\lambda + \lambda_\mu \lambda_{\min}$
    end if
  end if
else
  $r_i := \hat{\rho}$
end if
$C_i := \mathcal{A}^{\mathrm{leak}}(\mathcal{V}_i, r_i, \lambda)$ //(Def. 22)

Algorithm 9 Adaptive Agnostic Leakage Local Selection $\mathcal{A}^{\text{leak,agn,adap}}$

Parameters: Initial phase ι,
leak factor λ, minimal leak factor $\lambda_{\min}$, leak adaptation factor λ_μ,
drift compensation factor α, minimal drift compensation factor $\alpha_{\min}$, drift comp. adaptation factor α_μ
Input: View $\mathcal{V}_i = \{(s_j, h_j) | j \in [1, i]\}$
Output: Logical clock C_i

if $i > \iota$ **then**
  $r_i := R_{i-1}(h_i)$
  if $s_i - c_i^- > 0$ **then**
    $r_i := R_{i-1}(h_i) - \alpha(s_i - c_i^-)$
    $\lambda := (1 - \lambda_\mu)\lambda + \lambda_\mu \lambda_{\min}$
    $\alpha := (1 - \alpha_\mu)\alpha + \alpha_\mu \alpha_{\min}$
  end if
else
  $r_i := \hat{\rho}$
end if
$C_i := \mathcal{A}^{\text{leak}}(\mathcal{V}_i, r_i, \lambda)$ //(Def. 22)

Algorithm 10 Gradient CSA

Input: View $\mathcal{V}_i = \{(h_j, s_j) | j \leq i\}$
Output: Logical clock C_i
Parameters: Averaging-window size κ, initial phase ι
State: $S_{\text{loc}} := S_{\text{ref}} := D_{\text{loc}} := D_{\text{ref}} := 0$

// update predictor
$S_{\text{loc}} := S_{\text{loc}} + h_i$
$S_{\text{ref}} := S_{\text{ref}} + s_i$
if $i > \iota$ **then**
  $D_{\text{loc}} := (\kappa - 1)/\kappa * D_{\text{loc}} - 1/\kappa * S_{\text{loc}}/(i - 1)$
  $D_{\text{ref}} := (\kappa - 1)/\kappa * D_{\text{ref}} - 1/\kappa * S_{\text{ref}}/(i - 1)$
end if

// predict
if $D_{loc} > 0$ **then**
  $C_i(H(t)) := S_{\text{ref}}/i + (H(t) - S_{\text{loc}}/i) * D_{\text{loc}}/D_{\text{ref}}$
else
  $C_i(H(t)) := S_{\text{ref}}/i + (H(t) - S_{\text{loc}}/i)$
end if

Bibliography

[ADL+98] G. Asada, M. Dong, T. Lin, F. Newberg, G. Pottie, W. Kaiser, and H. Marcy. Wireless integrated network sensors: Low power systems on a chip. In *Proceedings of the European Solid States Conference*, 1998.

[AGR03] Kostas G. Anagnostakis, Michael Greenwald, and Raphael S. Ryger. cing: Measuring network-internal delays using only existing infrastructure. In *IEEE INFOCOM*, 2003.

[AHR93] Hagit Attiya, Amir Herzberg, and Sergio Rajsbaum. Optimal clock synchronization under different delay assumptions. In *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC 93)*, 1993.

[AP98] Emmanuelle Anceaume and Isabelle Puaut. Performance evaluation of clock synchronization algorithms. Technical Report 3526, Institut National de Recherche en Informatique et en Automatique, IRISA, Campus universitaire de Beaulieu, 35042 Rennes, France, October 1998.

[Arv94] K. Arvind. Probabilistic clock synchronization in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):474–487, May 1994.

[ASSC02] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: A survey. *Computer Networks (Elsevier)*, 38(4):393–422, March 2002.

[Bar03] Barix. Exstreamer wireless. `http://www.barix.com`, 2003.

[BD99] Eli Brandt and Roger B. Dannenberg. Time in distributed real-time systems. In *International Computer Music Conference (ICMC)*, 1999.

[BD01] Philipp Blum and Georg Dickmann. Verfahren zur Synchronisation in Netzwerken. International Patent Application (PCT) WO 03/047134 A2, November 2001.

[BD03] Philipp Blum and Georg Dickmann. Precise delay measurements in wired and wireless local area networks. Technical Report 175, Computer Enigneering and Networks Lab, D-ITET, ETH, 8092 Zurich, Switzerland, July 2003.

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

[BHHN00] Boaz Barak, Shai Halevi, Amir Herzberg, and Dalit Naor. Clock synchronization with faults and recoveries. In *Proceedings of the* 19^{th} *Ann. ACM Symp. on Principles of Distributed Computing*, pages 133–142, July 2000.

[BLTZ03] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. PISA – a platform- and programming-language-independent interface for search algorithms. In *Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632/2003 of *LNCS*, pages 494 – 508. Springer-Verlag Heidelberg, 2003.

[BMT04] Philipp Blum, Lennart Meier, and Lothar Thiele. Improved interval-based clock synchronization in sensor networks. In *Proceedings of the Information Processing in Sensor Networks (IPSN)*, 2004.

[Bri03] BridgCo. Wireless loudspeaker product information. `http://www.bridgeco.net/products/...` `...loudspeaker/wls.shtml`, 2003.

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

[BT04] Philipp Blum and Lothar Thiele. Trace-based evaluation of clock-synchronization algorithms for wireless loudspeakers. In *2nd Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia'04)*, pages 7–12, 2004.

[BW01] Saad Biaz and Jennifer L. Welch. Closed form bounds for clock synchronization under simple uncertainty assumptions. *Information Processing Letters*, 80(3):151–157, 2001.

[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.

[dAB94] Marcelo Morales de Azevedo and Douglas M. Blough. Fault-tolerant clock synchronization for distributed systems with high message delay variation. In *IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, pages 268–277, 1994.

[Dev03] Slim Devices. The wireless MP3 player for your stereo. `http://www.slimdevices.com/`, 2003.

[DHSS95] Danny Dolev, Joseph Y. Halpern, Barbara Simons, and Ray Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM*, 42(1):143–185, 1995.

[Dic04] Georg Dickmann. Synchronization in local area networks for multimedia distribution. Technical report, BridgeCo AG, Ringstrasse 14, CH-8600 Dübendorf, 2004.

[DRS94] Danny Dolev, Rüdiger Reischuk, and Ray Strong. Observable clock synchronization. In *Proceedings of the 13th ACM Symp. on Principles of Distributed Computing (PODC'94)*, pages 284–293, 1994.

[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.

[Edi02] John C. Edison. IEEE standard for a precision clock synchronization protocol for networked measurement and control systems. IEEE Std. 1588-2002, 2002.

[EE01] Jeremy Elson and Deborah Estrin. Time synchronization for wireless sensor networks. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, pages 1965–1970, April 2001.

[EGE02] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, December 2002.

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

[FC] The FlexRay Consortium. The communication systems for advanced automotive control applications. `http://www.flexray.com/`.

[FC95] Christof Fetzer and Flaviu Cristian. An optimal internal clock synchronization algorithm. In *Compass '95: 10th Annual Conference on Computer Assurance*, pages 187–196, 1995.

[Fie04] Ulrich Fiedler. Circuit emulation over gigabit-ethernet in MANs. submitted to IWQoS'04, February 2004.

[FLO02] Dominique Fober, Stephane Letz, and Yann Orlarey. Clock skew compensation over a high latency network. In *Proceedings of the International Computer Music Conference*, pages 548–552, 2002.

[FOL01] Dominique Fober, Yann Orlarey, and Stephane Letz. Real time musical events streaming over internet. In *1st International Conference on WEB Delivering of Music (WEDELMUSIC'01)*, 2001.

[Gal03] Peter L. Galison. *Einstein's Clocks and Poincare's Maps: Empires of Time*. W.W. Norton & Company, 2003.

[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 *Proceedings of the International Conference on Computer Design ICCD*, September 2002.

[GKAS03] Saurabh Ganeriwal, Ram Kumar, Sachin Adlakha, and Mani Srivastava. Network-wide time synchronization in sensor networks. Technical report, UCLA NESL, 2003.

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

[Gmb] Robert Bosch GmbH. Time triggered communication on CAN. `http://www.can.bosch.com/content/TT_CAN.html`.

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

[HL02] Lifei Huang and Ten-Hwang Lai. On the scalability of IEEE 802.11 ad hoc networks. In *Proceedings of the 3rd ACM International Symposium on Mobile Ad-Hoc Networking and Computing (MOBIHOC)*, pages 172–182, 2002.

[HMFH02] F. Hartwich, B. Müller, T. Führer, and R. Hugel. Timing in the TTCAN network. In *Proceedings of the 8th International CAN Conference (iCC'02)*, 2002.

[HMM85] Joseph Y. Halpern, Nimrod Megiddo, and Ashfaq A. Munshi. Optimal precision in the presence of uncertainty. *Journal of Complexity*, 1(2):170–196, 1985.

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

[HS03] An-swol Hu and Sergio D. Servetto. Asymptotically optimal time synchronization in dense sensor networks. In *2nd ACM International Conference on Wireless Sensor Networks and Applications*, pages 1–10, 2003.

[HWLC97] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global Positioning System: Theory and Practice, 4th ed.* Springer, 1997.

[IT00] Yutaka Ishibashi and Shuji Tasaka. A comparative survey of synchronization algorithms for continuous media in network environments. In *Proceedings of the 25th Annual IEEE Conference on Local Computer Networks (LCN)*, pages 337–348, 2000.

[KEES03] Richard Karp, Jeremy Elson, Deborah Estrin, and Scott Shenker. Optimal and global time synchronization in sensornets. Technical Report 0012, CENS, April 2003.

[KG94] Hermann Kopetz and Günter Grünsteidl. TTP – a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, January 1994.

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

[LGX00] Michael D. Lemmon, Joydeep Ganguly, and Lucia Xia. Model-based clock synchronization in networks with drifting clocks. In *Proceedings of 2000 Pacific Rim International Symposium on Dependable Computing*, December 2000.

[LKW03] Rainer Lienhart, Igor Kozintsev, and Stefan Wehr. Universal synchronization scheme for distributed audio-video capture in heterogeneous computing platforms. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 263–266. ACM Press, 2003.

[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.

[LMC99] Cheng Liao, Margaret Martonosi, and Douglas W. Clark. Experience with an adaptive globally-synchronizing clock algorithm. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 106–114, 1999.

[LS02] Nikolaos Laoutaris and Ioannis Stavrakakis. Intrastream synchronization for continuous media streams: A survey of playout schedulers. *IEEE Network Magazine*, 16(3):30–40, May 2002.

[Mar84] Keith Marzullo. Maintaining the time in a distributed system: An example of a loosely-coupled distributed service. Ph.D. thesis, Dept. of Electrical Engineering, Stanford University, 1984.

[Mat93] Friedemann Mattern. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing*, pages 423–434, 1993.

[MB03] Men Muheim and Philipp Blum. On the performance of clock synchronization algorithms for a distributed commodity audio system. In *Proceedings of the 114th Convention of the Audio Engineering Society (AES)*, March 2003.

[MBSP02] Reinhard Maier, Günther Bauer, Georg Stöger, and Stefan Poledna. Time-triggered architecture: A consitent computing platform. *IEEE Micro*, 22(5):36–45, 2002.

[MBT04] Lennart Meier, Philipp Blum, and Lothar Thiele. Interval synchronization of drift-constraint clocks in ad-hoc sensor networks. In *Proceedings of the fifth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'04)*, 2004.

[MDG01] Jörg Micheel, Stephen Donnelly, and Ian Graham. Precision timestamping of network packets. In *Proceedings of the First ACM SIGCOMM Workshop on Internet Measurement*, 2001.

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

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

[Mil93] David L. Mills. Precision synchronization of computer network clocks. *ACM Computer Communications Review*, 24(2):28–43, April 1993.

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

[MKSL04a] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. Flooding time synchronization in wireless sensor networks. In *Proceddings of the ACM SenSys*, 2004.

[MKSL04b] Miklos Maroti, Branislav Kusy, Gyula Simon, and Akos Ledeczi. The flooding time synchronization protocol. Technical Report ISIS-04-501, Institute for Software Integrated Systems, Vanderbilt University, Nashville Tennessee, 2004.

[MO83] Keith Marzullo and Susan Owicki. Maintaining the time in a distributed system. In *Proceedings of the second annual ACM symposium on Principles of distributed computing (PODC'83)*, pages 295–305. ACM Press, 1983.

[Mor85] Carroll Morgan. Global and logical time in distributed algorithms. *Information Processing Letters*, 20(4):189–194, May 1985.

[Mos04] Walter S. Mossberg. Gadget can send music on PC around a house without lots of wires. `http://ptech.wsj.com/archive/...` `...ptech-20040115.html`, 2004.

[MR03] Sayan Mitra and Jesse Rabek. Power efficient clustering for clock synchronization in dynamic multi-hop networks. unpublished, `http://theory.lcs.mit.edu/~mitras/courses/...` `...6829/project/final_report.ps`, 2003.

[MST99] Sue B. Moon, Paul Skelly, and Donald F. Towsley. Estimation and removal of clock skew from network delay measurements. In *Proceedings of the 1999 IEEE INFOCOM*, pages 227–234, 1999.

[Mur03] Charles J. Murray. Volkswagen exit puts fate of TTA body in doubt. EETimes: `http://www.eetimes.com`, August 2003.

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

[OPS99] Rafail Ostrovsky and Boaz Patt-Shamir. Optimal and efficient clock synchronization under drifting clocks. In *Proceedings of the*

Symposium on Principles of Distributed Computing (PODC'99), pages 3–12, 1999.

[OS94] A. Olson and K. G. Shin. Probabilistic clock synchronization in large distributed systems. *IEEE Transactions on Computers*, 43(9):1106–1112, 1994.

[Pax97] Vern Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, 1997.

[Pax98] Vern Paxson. On calibrating measurements of packet transit times. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/PERFORMANCE)*, pages 11–21, 1998.

[Phi03] Philips. Streamium. `http://www.streamium.com`, 2003.

[PSR94] Boaz Patt-Shamir and Sergio Rajsbaum. A theory of clock synchronization. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Montreal , Canada*, 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.

[PV01] Attila Pasztor and Darryl Veitch. A precision infrastructure for active probing. In *Proceedings of the Workshop on Passive and Active Measurement (PAM)*, 2001.

[Röm01] Kay Römer. Time synchronization in ad hoc networks. In *Proceedings of the ACM Symposium on Mobile Ad-Hoc Networking and Computing (MobiHoc'01)*, October 2001.

[Röm03a] Kay Römer. Temporal message ordering in wireless sensor networks. In *Proceedings of the IFIP Mediterranean Workshop on Ad-Hoc Networks*, pages 131–142, June 2003.

[Röm03b] Kay Römer. Time and location in sensor networks. In *Proceedings of the GI/ITG Workshop on Sensor Networks*, pages 57–60, July 2003.

[SBM+04] Janos Sallai, György Balogh, Miklos Maroti, Gyula Simon, and Akos Ledeczi. Acoustic ranging in resource constrained sensor networks. Technical Report ISIS-04-504, Institute for Software Integrated Systems, Vanderbilt University, Nashville Tennessee, 2004.

[SC90] F. Schmuck and F. Cristian. Continuous clock amortization need not affect the precision of a clock synchronization algorithm. In *Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing (PODC'90)*, pages 133–143, 1990.

[Sch97] Klaus Schossmaier. An interval-based framework for clock rate synchronization. In *Proceedings of the Symposion on Principles of Distributed Computing (PODC'97)*, pages 169–178, 1997.

[Sch98] Klaus Schossmaier. *Interval-based Clock State and Rate Synchronization*. PhD thesis, Technische Universität Wien, 1998.

[Sch03] Jürgen Schwager. Real-Time-Ethernet in der Industrieautomation. `http://www.real-time-ethernet.de`, 2003.

[SLWL90] Barbara Simons, Jennifer Lundelius-Welch, and Nancy Lynch. An overview of clock synchronization. In *Fault-Tolerant Distributed Computing (Asilomar Workshop 1986)*, number 448 in LNCS, pages 84–96, 1990.

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

[Ste90] Ralf Steinmetz. Synchronization properties in multimedia systems. *IEEE Journal on Selected Areas in Communications*, 8(3):401–412, April 1990.

[Str96] Hovey R. Strong. System for decoupling clock amortization from clock synchronization. United States Patent 5530846, International Business Machines Corporation (IBM), 1996.

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

[SW93] Hovey R. Strong and Edward L. Wimmers. Probabilistic anonymous clock synchronization and apparatus for synchronizing a local time scale with a reference time scale. United States Patent 5689688, International Business Machines Corporation (IBM), 1993.

[SW99] Klaus Schossmaier and Bettina Weiss. An algorithm for fault-tolerant clock state & rate synchronization. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems (SRDS)*, 1999.

[Tan96] Andrew S. Tanenbaum. *Computer Networks, third edition*. Prentice Hall, 1996.

[Tro94] Gregory D. Troxel. *Time Surveying: Clock Synchronization over Packet Networks*. PhD thesis, Massachusetts Institute of Technology, 1994.

[VCR93] Paulo Verissimo, Antonio Casimiro, and Luis Rodrigues. Using atomic broadcast to implement a posteriori agreement for clock synchronization. In *Proceedings of the Symposium on Reliable Distributed Systems*, pages 115–124, 1993.

[VG01] Martin Vetterli and Thomas Gross. National center of competence in research on mobile information and communication systems (NCCR MICS). Project webpage: `http://www.terminodes.org`, 2001.

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

[VR92] Paulo Verissimo and Luis Rodrigues. A posteriori agreement for faul-tolerant clock synchronization on broadcast networks. In *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS'92)*, pages 527–536, 1992.

[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.

[ZLT02] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Proceedings of the EUROGEN2001 Conference, Athens, Greece, September 19-21, 2001*, pages 95–100, 2002.

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

Curriculum Vitae

Date of Birth:

March 2, 1974

Place of Birth:

Beromünster, Kt. Luzern

Practical Experience:

2001–2004	Research and teaching assistant at the Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology Zürich (ETHZ)
1998–2000	Development engineer at Schindler Aufzüge AG in Ebikon, Kt. Luzern

Education:

1993–1998	Studies in microengineering at the Swiss Federal Institute of Technology Lausanne (EPFL); graduated as Ing. dipl. en Microtechnique.
1981-1993	Primary and secondary school, Beromünster, Kt. Luzern