

TIK-Report No. 187

Precise and Low-Jitter Wireless Time Synchronization

Philipp Blum¹ and Lothar Thiele
Department Information Technology and Electrical Engineering
Swiss Federal Institute of Technology Zurich (ETH)
Zurich, Switzerland
{blum, thiele}@tik.ee.ethz.ch

December 22, 2003

¹The author has received funding from the Swiss Commission for Technology and Innovation (KTI), Effingerstrasse 27, 3003 Bern, Switzerland, Grant number 4957.2

Abstract

In this paper, we are describing methods to perform time synchronization in a network of nodes that use wireless and ad-hoc communication, are decentralized, and have limited computing capabilities. Several new results are described. *Algorithms:* A new synchronization algorithm (LSA – Local Selection Algorithms) is presented that achieves low-microsecond precision and jitter, that is robust under instable connectivity and variable load conditions, and that requires only uni-directional communication. A variant of the algorithm (LSDC) is presented that dynamically compensates clock drift. *Formal Validation:* The achievable jitter and precision depend on the synchronization-message arrival-pattern and the synchronization-message delays. We propose a new analysis of clock synchronization algorithms under unknown pattern and delays, based on safety and liveness properties. LSA and LSDC are safe and live. *Experimental Validation:* Simulation results show that LSDC achieves $10\mu\text{s}$ precision on 802.11b wireless LAN in ad-hoc mode under variable load conditions.

Chapter 1

Introduction and Related Work

1.1 Precise and Low-Jitter Wireless Time Synchronization

Recently, there has been a growing interest in networks of nodes that use wireless communication, are decentralized, have limited computing capabilities and are operated under stringent power constraints, see for example [1, 5]. In this paper, we are describing new methods to perform a precise and low-jitter time synchronization in those systems. By *precise* we understand that the time offset between nodes in the network is small. A precise synchronization among nodes allows the coordination of activities on different nodes. *Jitter* refers to the ability of nodes to measure the exact length of a real-time interval. By *accurate* synchronization we understand the time of nodes in the network is close to real time which is useful for interaction with the wider world, e.g. in log files. Often, accuracy is less critical than precision and jitter.

There are many examples of application domains where a precise and low-jitter time synchronization is of major importance. Wireless sensor and actuator networks contain a large population of small-scale nodes which are closely coupled to their environment. One key element is the reading of sensor inputs and associating a time value to them. This way, the network of nodes can provide data fusion by combining various sensor inputs. Real time is used to relate activities (in the case of actors) or events (in the case of sensors) to the physical world.

Many applications rely on a precise and low-jitter timing synchronization on the order of a few microseconds. Examples are switching off the radio between two brief message transmissions in order to save energy [5], object tracking (acceleration and velocity) by performing a fusion of sensor readings from different, spatially distributed sensors, distributed beam forming and signal processing using sensor arrays [22], measuring the time of flight of sound for localizing its source [9]. Finally, recent distributed applications in the domain of digital multi-channel audio, e.g. wireless loudspeakers, require clock synchronization between nodes in the order of $10\mu\text{s}$.

In case of a dynamic and ad-hoc connectivity, nodes may physically move and we are faced with an unpredictable connectivity. This not only concerns the quality of transmission because of the changing radio propagation (delay, message loss, communication failure), but also the presence of a particular communication link at all. Mobile applications in particular impose a whole new set of requirements to the problem of time synchronization, see for example [7]. We are considering clock synchronization

in wireless networks under the following objectives: Achieve precise synchronization among nodes on the order of a few microseconds. Achieve low-jitter relative to real time on the order of a few microseconds. Achieve synchronization to real time on the order of milliseconds.

In this paper we describe a simple scenario which can be regarded to be a building block for more complex synchronization tasks. Communication is asynchronous and ad-hoc, messages may get lost or arbitrarily delayed. Clock drift can vary because of changes in battery power and temperature and, to a lesser degree, because of oscillator aging. Computation and Communication overhead must be small due to limited capabilities of the nodes and stringent power constraints. Ideally, time synchronization uses the communication pattern generated by the main application in the network and does not generate traffic on its own behalf.

1.2 Related Work

Several probabilistic clock synchronization algorithms have been proposed to achieve a good accuracy in the asynchronous communication model. These methods are usually based on the exchange of clock information and additional means to reduce the effects of nondeterminism in message delivery and processing. A prominent example is the network time protocol (NTP) by Mills, see [16]. The protocol uses a hierarchy of servers and is known for its scalability and robustness against server failures and malicious attacks. NTP is not designed for microsecond accuracy and requires configuration which is contrary to the ad-hoc paradigm. Like NTP, the algorithm by Cristian [6] and other similar algorithms [3, 13] use a client-server approach to read reference time repeatedly. These algorithms continue reading the reference time until this succeeds with a specified precision bound.

[6] achieves a precision on the order of 1ms using 4 messages per minute. Similar results have been obtained in [3]. [13] achieves $1\mu\text{s}$ using highly deterministic Myrinet communication. A producer-consumer approach is taken by Arvind in [4]: With unidirectional communication only, a precision in the order of 1ms is achieved. This approach requires that statistical properties of the message delays are known and remain constant.

There have been several results on time synchronization dedicated to wireless applications. Römer [20] achieves a 1ms accuracy by creating an instantaneous time scale. In particular, he needs little overhead and uses unidirectional links only. The CesiumSpray approach [21] relies on the availability of a broadcast medium and uses an external time standard. The clients of a broadcast domain are synchronized relative to each other. Hill [11] achieves a time accuracy of $2\mu\text{s}$ but relies heavily on dedicated hardware and a tight integration of his synchronization method with lower layers of the communication protocol. Mock et. al [17] achieve an accuracy of about $150\mu\text{s}$ using commercial off-the-shelf technology (802.11b Wireless LAN).

For broadcast networks, Elson et.al. propose in [8] a scheme called RBS that eliminates the medium-access time from the synchronization path. All client nodes within a single physical broadcast domain are synchronized with each other by means of a centralized server node which provides timing messages. After a certain number of timing messages have been sent, the information gathered by the clients is unified and information about the relative clock frequencies and the timing offset is calculated and redistributed to the clients. The approach requires broadcast, bidirectional communication and stable connectivity. [8] reports that the RBS algorithm achieves a precision in

the order of $10\mu\text{s}$ using 802.11b Wireless LAN.

We identified the following deficiencies of most known algorithms: *Scalability*: Algorithms that use round-trip transactions with a common source of synchronization do not scale well. Every new node that joins the system adds to the load of the source of synchronization. Algorithms that use only uni-directional communication can make use of broadcast communication for scalability reasons. *Robustness against variable load conditions*: The capacity of wireless networks is limited. Therefore, such networks are often operated close to saturation. This means that statistical properties like average or median delay vary largely. It also means that a-priori symmetric channels become biased. This is a problem for algorithms that assume symmetry or use the average delay for synchronization.

1.3 New Results

In this paper we present the Local Selection Algorithm (LSA) and its extension, the Local Selection with Drift Compensation (LSDC) algorithm that both tackle the deficiencies mentioned above. LSA and LSDC use uni-directional communication only, as does Arvind’s algorithm [4]. Therefore, LSA and LSDC may use broadcast for scalability reasons, but they do not require it. Different from [4], the new algorithms do not average many observations, but compute a lower bound on real time. As will be shown in the remaining sections, the new algorithms are sensitive to the *minimal* message delay rather than to the average or median delay, which leads to increased robustness against heavy and variable load conditions.

Formal Analysis: Unknown system properties like non-deterministic message delay, synchronization message pattern and variable clock drift do not allow to apply bounds on the achievable accuracy as proposed by several authors [12, 10, 14, 19, 18]. We propose an alternative analysis of clock synchronization algorithms under unknown system specifications in terms of message delay and message pattern: Instead of bounds on the achievable accuracy, we propose a definition for two properties that describe “good” algorithms. *Safe synchronization* never degrades the accuracy of a synchronized clock. *Live synchronization* eventually improves the accuracy. We show that the new algorithms are safe and live while most known algorithms fail to meet the safety- or the liveness-property. We also show, that in the cases of LSA and LSDC, the precision is not larger than the jitter.

New algorithms: We propose two new algorithms based on the computation of lower bounds on real time. LSA needs no parameterization, and has a very small computing overhead. LSDC improves LSA in that it uses a heuristics to dynamically compensate clock drift.

Experimental validation: We evaluate the new algorithms based on real delay and clock traces, obtained in a 802.11b Wireless LAN in ad-hoc mode and standard Linux PCs. Various load scenarios are tested in order to validate the robustness of the algorithms, for example cross traffic in the same WLAN domain and the use of the same interface for other communications. The algorithm with drift compensation reliably achieves a precision of $10\mu\text{s}$ under variable heavy-load conditions.

Chapter 2

Problem Definition

In this section we formally define the system (nodes, clocks, communication) and the requirements for clock synchronization.

2.1 System Model

We consider a system S of nodes that can receive synchronization messages from at least one *synchronization source* that has access to real time t , e.g. by means of a GPS receiver. The nodes in S all have an embedded local clock h that has an arbitrary offset and bounded drift relative to real time. The nodes run a clock synchronization algorithm to compute an estimate of real time as a function of the local clock and local state information, e.g. node p estimates real time as $T^p(h^p(t))$ when it actually is t . See Figure 2.1 for an illustration. If we talk about an anonymous node, we omit the superscript. The performance of the algorithm is characterized by the synchronization error:

Definition 2.1 (Synchronization error) *The synchronization error of node at real time t is the difference between the node's estimate of real time T and real time, i.e.*

$$\epsilon(t) = T(h(t)) - t. \quad (2.1)$$

Based on the notion of synchronization error, we define the *accuracy* and the *jitter* of a node.

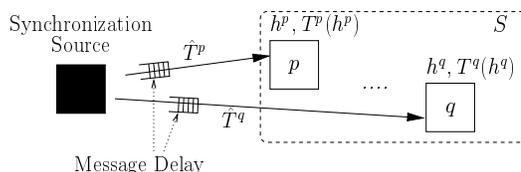


Figure 2.1: Every node (e.g. p and q) in the system S may receive a messages with a time stamp \hat{T} which denotes the real time t when the message was sent by the synchronization source. Nodes have an embedded hardware clock h and compute an estimate of the current real time T .

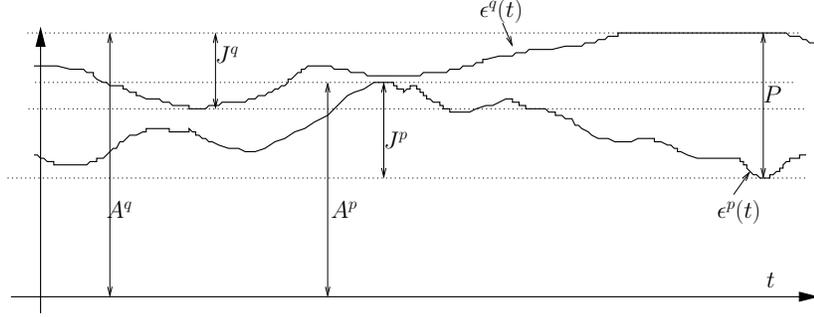


Figure 2.2: Accuracy and jitter of the nodes p and q . The precision of the system S can be much smaller than the accuracy of any node in S if the system does not contain the source of synchronization and all nodes have a similar offset to real time t . The figure shows a generic scenario and does not refer to the synchronization achieved by LSA.

Definition 2.2 (Accuracy) A node is said to have an accuracy of A , if the absolute value of its synchronization error is bounded by A at any time t after the initialization time t_I :

$$\forall t \geq t_I : |\varepsilon(t)| \leq A \quad (2.2)$$

Definition 2.3 (Jitter) A node is said to have a jitter of J if for any time interval starting after initialization time t_I , the difference between the measured length $T(h(t_2)) - T(h(t_1))$ and the real length $t_2 - t_1$ of this interval is bounded by J :

$$\forall t_2 \geq t_1 \geq t_I : |\varepsilon(t_2) - \varepsilon(t_1)| \leq J \quad (2.3)$$

In contrast to the node based properties accuracy and jitter, the *precision* is defined for the whole system.

Definition 2.4 (Precision) A system S is said to have a precision of P if, after initialization time t_I , the maximal distance between any two nodes' estimate of real time is bounded by P :

$$\forall t \geq t_I : \max \{ |\varepsilon^p(t) - \varepsilon^q(t)| : \forall p, q \in S \} \leq P \quad (2.4)$$

In Figure 2.2 the relation between accuracy, jitter and precision is illustrated. Both jitter and precision can be much smaller than the accuracy. Note that the system S does not contain the source of synchronization. This is similar to the situation when using the RBS scheme [8]: The node that transmits synchronization beacons cannot be synchronized to the receivers of this beacon, unless the node contains a second network interface on which the beacon is received with the same delay as other nodes experience it.

2.2 Communication Model

Every node in the system receives synchronization messages from a synchronization source. The nodes do not utilize any additional communication for synchronization, neither messages back to the synchronization source nor messages to any other nodes in the system.

Every node receives an infinite sequence of synchronization messages, the index i is used to denote these events. As illustrated by Figure 2.3, the i -th synchronization message is received at real time t_i and contains the time stamp \hat{T}_i representing real time when the message was sent. The delay of the message is $d_i = t_i - \hat{T}_i$. We assume asynchronous communication, i.e. there exists no a-priori upper bound on the message delay. The minimal delay is $d_{\min} = \min \{d_i : i \in \mathbf{N}\}$. It is not known and can not be determined by the nodes, using only uni-directional communication.

2.3 Clock Model

Each node in the system S has an embedded hardware clock. The value of the hardware clock at time t is denoted as its local time $h(t)$. Based on the speed $dh(t)/dt$ of the hardware clock we can define its drift at time t as

$$\rho(t) = dh(t)/dt - 1. \quad (2.5)$$

In addition, we will also need the rate of change of the drift. To this end,

$$\vartheta(t) = d\rho(t)/dt \quad (2.6)$$

denotes the drift variation. In order to simplify the notation, we denote the properties of the clock at event i as $h_i = h(t_i)$ (local time), $\rho_i = \rho(t_i)$ (drift) and $\vartheta_i = \vartheta(t_i)$ (drift variation). Note that at the occurrence of some event i , a node does in general not know the real time t_i , the drift of its own clock ρ_i , or its variation ϑ_i . The only information the node can get from its hardware clock is the local time h_i .

Assumption 2.1 (Bounded Clock Drift) *The clocks have a minimal and maximal drift. The associated constraint can be formulated as*

$$|\rho(t)| \leq \rho_{\max} \quad \forall t. \quad (2.7)$$

Assumption 2.2 (Oscillator stability) *We will also assume that the drift variation is bounded as follows:*

$$|\vartheta(t)| \leq \vartheta_{\max} \quad \forall t. \quad (2.8)$$

These bounds represent a-priori assumptions about the properties of the hardware clock.

2.4 Node Model

The nodes maintain a local state, where they collect certain information from messages received in the past and corresponding readings of their embedded hardware clock. Local state variables are marked by the subscript $_{LS}$. Real time t_{LS} is not part of the local state but helps in the analysis: it represents the real time when the local state was last modified.

At any time t , a node may compute an estimate $T(h(t))$ of current real time from the current reading of its hardware clock h and local state information.

The nodes use a clock synchronization algorithm to update the local state upon reception of a synchronization message at real time t_i . The indices $i-, i+$ are used to refer to variables and properties immediately before and after the i -th state update respectively.

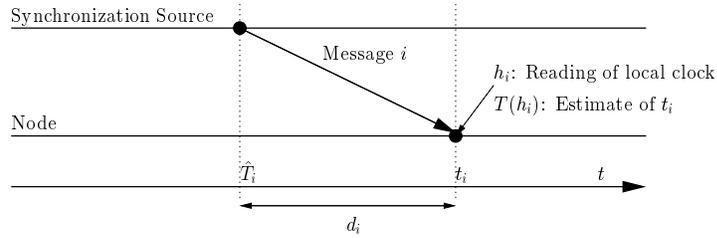


Figure 2.3: Sequence diagram of the i -th message sent from the synchronization source to a node. It is received at real time t_i and contains the time stamp \hat{T}_i . At the message arrival, the local clock shows h_i and the node estimates real time as $T(h_i)$.

2.5 Discussion

In sensor networks, it is of utmost importance to accurately capture the length of a physical interval, thus a low jitter is required. If activities among several nodes have to be coordinated, precision is required. To cooperatively measure intervals using more than one node, low jitter and good precision is required, but not accuracy. Very often accuracy is not critical, as the coupling of the system with the wider world is rather loose. In summary, we require *Precise synchronization among the nodes*: For sensor data fusion and event ordering, a precision of a few microseconds shall be achieved. *Low jitter*: The accuracy of any observed interval relative to real time shall be on the order of a few microseconds. Only then sensor data can be interpreted correctly, e.g. to translate a time of flight in seconds to a distance in meters. *Accurate synchronization to real time*: An accuracy on the order of milliseconds is often sufficient, e.g. for writing log-files.

On the other hand, it is possible to achieve a synchronization with high accuracy if we use additional means to estimate the time offset, for example by providing a bidirectional link between a node and the synchronization source and using the corresponding round-trip delay.

Chapter 3

Local Selection Algorithm

In this section, we present and discuss the basic idea of the Local Selection Algorithm (LSA).

3.1 Basic Idea

In the related-work section, we identified *uni-directional* communication as a design goal for clock synchronization algorithms in large-scale sensor networks. Additionally, we argued for *robustness against heavy and quickly varying delay conditions*.

Unidirectional communication in the asynchronous model essentially means that we cannot know how long a message traveled between sender and receiver, only the causal order between related send and receive events can be determined. A node that receives synchronization messages can compute a lower bound on real time but no upper bound. As a consequence, this node cannot know anything about its accuracy or precision. Fortunately, in many asynchronous systems only few messages get excessively delayed or lost. Still, the average delay varies depending on the load in the communication channel and in the network interfaces of sender and receiver nodes. However, the minimal delay in wireless Ethernet networks remains fairly constant even under heavy load, though only a reduced number of messages experience the minimal delay. These two observations have led to the basic idea of clock synchronization based on the computation of a *lower bound on real time*: Upon every synchronization message arrival, LSA computes a lower bound on real time from the embedded hardware clock and local state. If the received timestamp indicates a tighter lower bound on real time, the local state is updated accordingly.

Even though RBS and our new algorithms tackle the same sort of problems, mainly to achieve a good precision, the fundamental ideas behind the two algorithms are quite different and may be combined favorably in future work: While RBS uses the observation that most communication delay uncertainty is due to medium access and that this can be removed from the synchronization path if the system allows physical broadcast, LSA uses the observation that every channel (be it broadcast or uni-cast) has a recurring minimal delay and that this minimal delay is much more stable under variable load conditions than the average delay or the median delay.

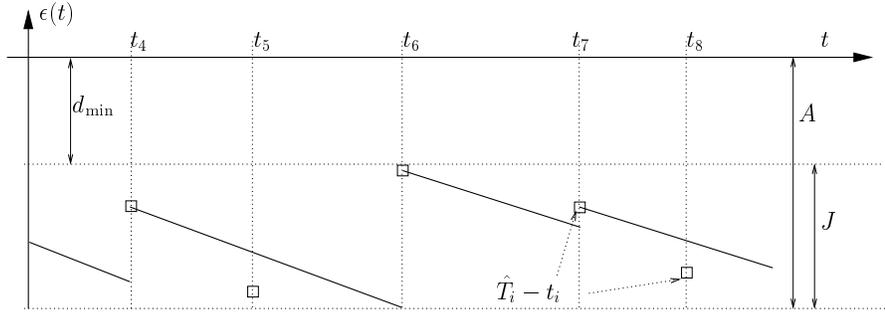


Figure 3.1: Synchronization error of the LSA. The boxes represent the error of the timestamps contained in the synchronization messages at their time of arrival. The synchronized Time T is a lower bound on real time t .

3.2 The Algorithm

The Local Selection Algorithm (LSA) maintains the local state variables h_{LS} and T_{LS} . The local state is initialized with $T_{LS} = 0$ and $h_{LS} = 0$.

The function $updateLowerBound(h)$ is executed with the current hardware-clock time h as input whenever an application demands an estimate of real time. The function delivers the synchronized time T as output:

Algorithm 1 $updateLowerBound(h)$

1: $T \leftarrow T_{LS} + \frac{1}{1+\rho_{max}} (h - h_{LS})$

Upon every synchronization message reception, the Local Selection Algorithm evaluates the received timestamp \hat{T} and updates the local state if the received timestamp is a better lower bound than the the local synchronized time T :

Algorithm 2 Local Selection Algorithm (LSA)

Require: Reference Timestamp \hat{T}_i received at h_i

1: $T_i \leftarrow updateLowerBound(h_i)$
2: **if** $\hat{T}_i > T_i$ **then**
3: $T_{LS} \leftarrow \hat{T}_i$
4: $h_{LS} \leftarrow h_i$
5: **end if**

Chapter 4

Analysis

In this section we show that LSA computes a lower bound on real time. We introduce the properties *safety* and *liveness* for clock synchronization algorithms and show that LSA is safe. It is also live under the recurring delay assumption that is formalized in this section. We finally discuss the relation between the accuracy, the jitter and the precision produced by LSA.

4.1 Lower Bound on Real Time

Figure 3.1 shows a sample trace of the Local Selection Algorithm. First we note that the synchronization error is always negative, as would be expected by an algorithm that computes a lower bound on real time. The algorithm updates the local state at t_4, t_6 and t_7 , but not at t_5 and t_8 . Because all synchronization messages are delayed by at least d_{\min} , the synchronization error immediately after the update is smaller than $-d_{\min}$. The synchronization error between updates decreases. It always does, as shows the following lemma:

Lemma 4.1 (Lower Bound on Real Time) *The function $\text{updateLowerBound}(h)$ computes a valid lower bound T on real time at hardware-clock time h if for the local state T_{LS} is a valid lower bound on real time at h_{LS} . If in addition T_{LS} is a tight bound, then T is tight also.*

Proof 4.1 *Let t be the real time at which the hardware clock shows h . By Equation 2.7 we know that evolved clock time is bounded: $(h - h_{LS}) \leq (1 + \rho_{\max})(t - t_{LS})$ and therefore $T \leq T_{LS} + \frac{1 + \rho_{\max}}{1 + \rho_{\max}}(t - t_{LS})$. By transformation we get $T - t \leq T_{LS} - t_{LS}$: if T_{LS} is a lower bound, i.e. $T_{LS} - t_{LS} \leq 0$, then also T is a lower bound, i.e. $T - t \leq 0$. T is tight if $T_{LS} = t_{LS}$ and the clock advances at its maximal speed in $[t_{LS}, t]$, i.e. $(h - h_{LS}) = (1 + \rho_{\max})(t - t_{LS})$.*

4.2 Safety and Liveness

As a consequence of Lundelius' and Lynch's fundamental result [14], we cannot hope to find deterministic analytical bounds on the accuracy and on the jitter LSA achieves in the asynchronous-communication model. It depends on the frequency of arrivals of fast synchronization messages and on the drift of the lower bound T . As we make

no assumptions on a particular pattern of synchronization messages, we also cannot provide probabilistic bounds as do [4] and [6] for their algorithms.

We propose a new analysis of clock synchronization algorithms, applicable in the asynchronous communication model and under unknown message patterns. The analysis consists of the properties *safety* and *liveness*. Though this is a common way to prove the correctness of algorithms, it is not obvious how to apply them to clock synchronization algorithms.

Definition 4.1 (Safety) *A clock synchronization algorithm is safe, if the absolute value of the synchronization error ϵ does not increase upon the reception of synchronization message i and the corresponding actions of the algorithm.*

$$\forall i : |\epsilon(t_{i+})| \leq |\epsilon(t_{i-})| \quad (4.1)$$

Theorem 4.1 (Safety) *The Local Selection Algorithm is safe.*

Proof 4.2 *The algorithm modifies the synchronization error whenever a received timestamp \hat{T}_i is larger than the local synchronized time T and therefore the synchronization error after the update is always larger than before. Since the synchronization message is delayed by at least d_{\min} , we know that $\hat{T} - t \leq -d_{\min}$, i.e. the synchronization error immediately after an update is always negative. If the synchronization error increases and still is always negative, it's absolute value must decrease, and therefore the algorithm is safe.*

But note that even an algorithm that does 'nothing' is safe as it does not decrease the synchronization error when receiving a message, but it does not improve it either. Therefore, we introduce the notion of liveness.

Definition 4.2 (Liveness) *A clock synchronization algorithm is live, if for every time t there is a later time t_i at which the algorithm achieves the same or a better absolute value of the synchronization error ϵ .*

$$\forall t : \exists i : (t_i > t) \wedge (|\epsilon(t_{i+})| \leq |\epsilon(t)|) \quad (4.2)$$

We can not show liveness of the LSA under every possible sequence of message delays. Therefore we make the assumption, that short delays occur repeatedly:

Assumption 4.1 (Recurring Delay) *Every delay that occurs is followed eventually by an equal or shorter delay.*

$$\forall i : \exists j > i : d_j \leq d_i \quad (4.3)$$

Theorem 4.2 (Liveness) *Under the Recurring Delay Assumption, the Local Selection Algorithm is live.*

Proof 4.3 *Assume that a local state update is performed at t_i : $T_{LS} = \hat{T}_i$, $h_{LS} = h_i$. The current estimate at t_j therefore is $T_j = \hat{T}_i + \frac{1}{1+\rho_{\max}}(h_j - h_i)$. Using Lemma 4.1 we get $T_j \leq \hat{T}_i + (t_j - t_i)$, and therefore $T_j \leq t_j - d_i$. Replacing t_j with $\hat{T}_j + d_j$, we get $T_j \leq \hat{T}_j + d_j - d_i$ and finally $T_j \leq \hat{T}_j$. Therefore the algorithm will update the local state and improve accuracy.*

It can be shown that known probabilistic algorithms are not safe. Reasons are averaging over several observations [4, 8, 15] or selecting synchronization messages to update the local state based on their round-trip delay [6, 2, 13]. Both techniques do not guarantee to improve the absolute accuracy, though it is very likely that they do. Lamport's Algorithm [12] is safe, but not live if the embedded hardware clock runs faster than real time.

4.3 Precision

Again referring to Figure 3.1, we see that the jitter J is by d_{\min} smaller than the accuracy A . If the nodes in the system have an equal minimal delay, then their offsets cancel each other. Such a favorable situation occurs in single hop networks, e.g. wired or wireless LANs and with nodes that have a comparable network interface. Only the sending node, i.e. the synchronization source is excluded from this set.

Assumption 4.2 (Equal Minimal Delay) *For every two nodes p and q in S , the received synchronization messages have the same minimal delay,*

$$\forall p, q \in S : d_{\min}^p = d_{\min}^q \quad (4.4)$$

Corollary 4.1 (Precision) *Under the Equal Minimal Delay Assumption, the precision of the system S , consisting of nodes that use the Local Selection Algorithm, equals at most the largest jitter of any node $p \in S$.*

$$P^S \leq \max \{J^p : \forall p \in S\} \quad (4.5)$$

This is different from the generic situation depicted in Figure 2.2. There, the precision is worse than the jitter of either node p and q .

Chapter 5

Local Selection with Drift Compensation

In this section we present a variant of the Local Selection Algorithm with drift compensation, in short LSDC. LSDC uses the same basic principle as LSA, but it dynamically estimates and compensates clock drift.

5.1 Drift Compensation

In the last section we have seen that the jitter J depends on the frequency of fast synchronization messages and clock drift. The algorithm cannot have an influence on the message frequency as long as we do not use bi-directional communication. However, a clock synchronization algorithm can try to estimate the current clock drift and compensate for it. Figures 3.1 and 5.1 explain the principle: The synchronization error of the variant with drift compensation (LSDC) remains stable for some time, while it quickly decreases in the case of LSA. The accuracy and the jitter are better in the case of LSDC than with LSA.

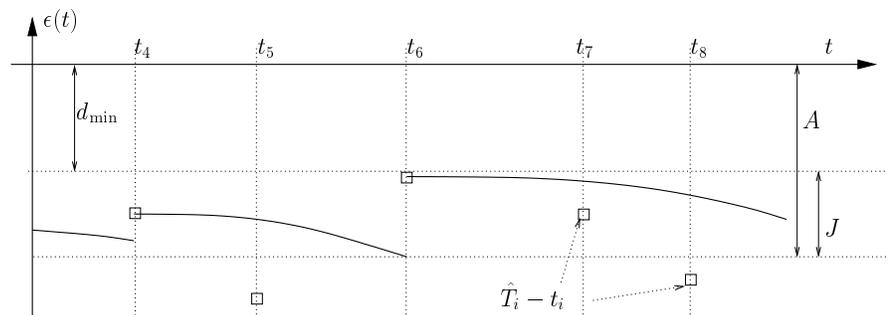


Figure 5.1: Synchronization error of the LSDC. The boxes represent the error of the timestamps contained in the synchronization messages at their time of arrival. LSDC achieves a better jitter and accuracy than LSA in Figure 3.1 due to drift compensation.

5.2 The Algorithm

The Local Selection Algorithm maintains the local state variables h_{LS} , T_{LS} , and R_{LS} . The local state is initialized with $T_{LS} = 0$, $h_{LS} = 0$, and $R_{LS} = \rho_{\max}$.

Whenever an application requires an estimate of real time, the function *updateLowerBound2*(h) is executed with the current hardware clock time h as input. The function delivers the synchronized time T as output:

Algorithm 3 *updateLowerBound2*(h)

```

1: if  $h < h_{LS} + \frac{(1+\rho_{\max})^2 - (1+R_{LS})^2}{2\vartheta_{\max}}$  then
2:    $T \leftarrow T_{LS} + \sqrt{\left(\frac{1+R_{LS}}{\vartheta_{\max}}\right)^2 + 2\frac{h-h_{LS}}{\vartheta_{\max}} - \frac{1+R_{LS}}{\vartheta_{\max}}}$ 
3: else
4:    $T \leftarrow T_{LS} + \frac{(\rho_{\max}-R_{LS})^2}{2\vartheta_{\max}(1+\rho_{\max})} + \frac{h-h_{LS}}{1+\rho_{\max}}$ 
5: end if

```

It can be shown that *updateLowerBound2*(h) delivers a lower bound on real time at hardware-clock time h if T_{LS} is a lower bound on real time at hardware-clock time h_{LS} , and R_{LS} is an upper bound on the clock drift ρ at hardware-clock time h_{LS} . The proof is similar to that of Lemma 4.1. But now, we know that by Equation 2.8, the elapsed hardware-clock time is bounded by $(h - h_{LS}) \leq (1 + R_{LS})(t - t_{LS}) + \frac{1}{2}\vartheta_{\max}(t - t_{LS})^2$.

Upon every synchronization message reception, the local selection algorithm with drift compensation (LSDC) evaluates the received timestamp \hat{T}_i and updates the local state, if the received information leads to a better lower bound. Whereas the setting $R_{LS} = \rho_{\max}$ is a valid choice, we will show in the next section that there is the possibility to estimate the current value of R_{LS} which leads to a better synchronization behaviour.

Algorithm 4 Local Selection Algorithm with Drift Compensation (LSDC)

Require: Reference Timestamp \hat{T}_i received at h_i

```

1:  $T_i \leftarrow \text{updateLowerBound2}(h_i)$ 
2: if  $\hat{T}_i > T_i$  then
3:    $R_{LS} \leftarrow \text{estimateDriftBound}(h_i, \hat{T}_i)$ 
4:    $T_{LS} \leftarrow \hat{T}_i$ 
5:    $h_{LS} \leftarrow h_i$ 
6: end if

```

5.3 Heuristic for Drift Compensation

The LSDC algorithm estimates the drift of the embedded hardware clock h . Assume the algorithm would know real times t_1 and t_2 and the associated clock times h_1 and h_2 . As illustrated by Figure 5.2, the algorithm could calculate the following bound on the clock drift at real time t :

$$1 + \rho(t_2) \leq \frac{h_2 - h_1}{t_2 - t_1} + \frac{1}{2}\vartheta_{\max}(t_2 - t_1). \quad (5.1)$$

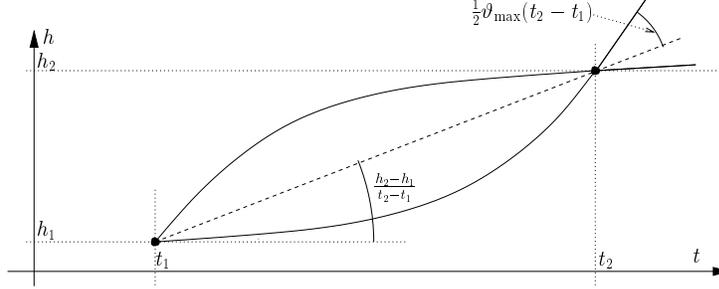


Figure 5.2: Illustration of the heuristic for drift estimation: If two pairs of real time t and clock time h are known, then clock drift can be bounded. The dashed line represents a clock with constant drift. The solid lines represent clock whose drift increases, resp. decreases with the maximal speed of ϑ_{\max} .

But of course, the algorithm cannot know t_1 and t_2 nor can it determine the length of the interval $t_2 - t_1$. At least, the length of the interval is bounded:

$$(T_2 - T_1) - J < (t_2 - t_1) < (T_2 - T_1) + J. \quad (5.2)$$

As the algorithm cannot know J , it is replaced by a parameter $\alpha > J$. If t_2 is replaced by a synchronization message arrival event that leads to an improvement in the real time bound t_i and t_1 is interpreted as the last local state update, then the following heuristic results:

Algorithm 5 *estimateDriftBound*(h_i, \hat{T}_i)

$$1: R_i \leftarrow \min \left(\rho_{\max}, \frac{h_i - h_{LS}}{\hat{T}_i - T_{LS} - \alpha} - 1 + \frac{1}{2} \vartheta_{\max} (\hat{T}_i - T_{LS} + \alpha) \right)$$

A large α leads to a high probability that $J < \alpha$ and therefore R_i is a valid upper bound on clock drift and in consequence T_i is a valid lower bound on real time. But on the other hand, a large α also leads to incomplete and slow drift compensation. A small α leads to fast drift compensation but at an increased risk of over-compensation, i.e. R_i is not a valid upper bound on drift and T is not a valid lower bound on real time for a certain bounded period. This dilemma can partly be resolved when more memory is available where we store some history of the local state. This way, one can consider a larger time interval in order to estimate the drift. Even a large α becomes small relative to $h_i - h_{LS}$ and $\hat{T}_i - T_{LS}$, and therefore efficient drift compensation is possible with only a small risk of overcompensation.

Chapter 6

Experimental Study

In this section we present the experimental evidence that indeed the Local Selection Algorithms can achieve precision and jitter in the order of microseconds using commercial-off-the-shelf (COTS) components under heavy and variable network-load conditions. The study consists of two parts: We measure and record *message-delay and clock-drift traces* of a COTS-system comprising Linux-PC nodes and a 802.11b Wireless LAN in ad-hoc mode under heavy and variable network-load conditions. We evaluate and compare the *precision* achieved by Local Selection Algorithm with Drift Compensation (LSDC) and the Reference Broadcast Synchronization (RBS) scheme [8]. We use Matlab implementations of the algorithms and apply them to the previously recorded delay and clock traces.

6.1 Experimental System Setup

The experimental setup is illustrated in Figure 6.1.

6.1.1 Network:

An IEEE 802.11b Wireless LAN in ad-hoc mode is used. The network comprises six nodes: The synchronization source m , the synchronization clients p and q and three additional nodes that generate load in the network.

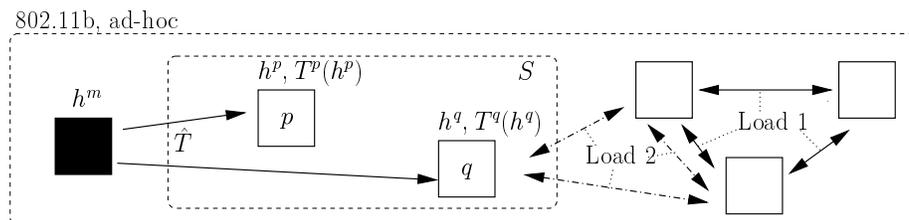


Figure 6.1: Experimental setup. Six nodes form a wireless 802.11b ad-hoc network. One node acts as source of synchronization and two other nodes synchronize to the source. Three other nodes generate additional communication load on the common medium. This creates different delay patterns for the synchronization messages.

6.1.2 Nodes:

All nodes are Linux PCs. p and m run at 500 MHz processor speed, q at 300 MHz. The nodes use the Timestamp-Counter-Register (TSC) of the Pentium processor as the embedded hardware clock. As it runs with the processor speed, it is normalized to $1\mu s$ units. The receive timestamps h_i are taken in Kernel-Space, in the interrupt routine of the network interface.

6.1.3 Synchronization Source:

The synchronization source uses broadcast messages to distribute timestamps \hat{T}_i to the clients every 20 milliseconds. The source has no access to real-time, instead its own local clock h^m is used as the reference clock. The parameters ρ_{\max} and ϑ_{\max} of the clients are doubled compared to the original values to account for the variations of the reference clock.

6.1.4 Load:

Traces have been recorded under three load conditions: No load. Sporadic file transfers between the three nodes that are not synchronization clients. And finally Sporadic file transfers between two nodes that are not synchronization clients and client q . See Figure 6.1 for an illustration.

6.1.5 Measurements:

To measure message delays and clock drift, we need to know reference time when the synchronization messages are received. To this purpose, nodes m , p and q are connected by their parallel ports on which we generate edges at a periodic intervals. The nodes latch the embedded clock whenever the edges on the parallel port trigger interrupts. The latched values are stored in a trace file. The accuracy this method was calibrated to be better than $5\mu s$.

6.2 Delay and Clock Traces

Figure 6.2 shows the message delay distribution of the synchronization messages in the three load scenarios. All scenarios are based on a sequence of approximately 36000 synchronization messages each, i.e. a stream of 20 synchronization messages per second during 30 minutes.

		$d_{\min} [\mu s]$	$d_{\text{median}} [\mu s]$	$d_{\text{average}} [\mu s]$	$d_{\max} [\mu s]$
No Load	Node p	974	992	1025	1108
	Node q	976	996	1029	1108
Load 1	Node p	974	1274	2110	11548
	Node q	981	1290	2127	11562

Table 6.1: Delay statistics for two different load scenarios. While the average and median delay varies, the minimal delay remains constant. The minimal delay is the same for both nodes.

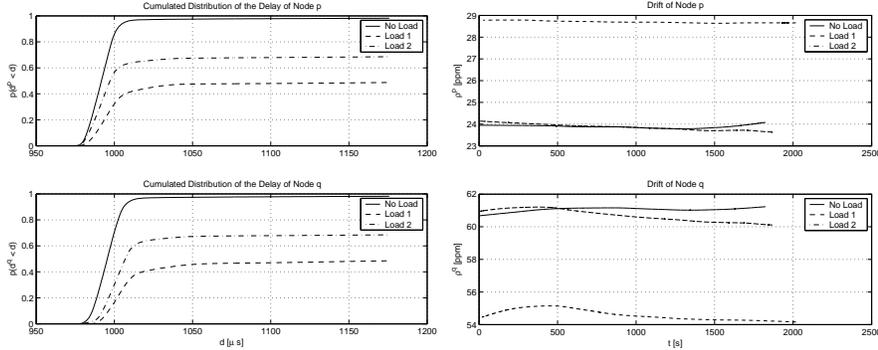


Figure 6.2: Left: Synchronization-message delay distributions. Both nodes have a similar minimal delay. The minimal delay remains more constant than the median delay under different load conditions. Right: Clock-drift traces. Clock drift is variable. Between the measurement with Load 2 and the other two measurements the computer had been switched off over-night.

The delay statistics are shown in Table 6.1. The results confirm that the minimal delay is constant in different load scenarios. The clock-drift traces show that clock drift varies about up to 1ppm in ten minutes. The traces also show that clock drift can be significantly different (several ppm) over a long time.

6.3 Matlab simulation of internal synchronization

The recorded traces were used as stimuli for Matlab implementations of the LSDC and RBS algorithms. The additional communication required for RBS (between nodes p and q) is implicit as all computations are done within one Matlab simulation on one computer. The LSDC uses *estimateDriftBound2* with $\alpha = 100\mu s$. Additionally a FIFO buffer of older state information has been used to improve drift estimation. The RBS clients used 1000 pairs of timestamps (h_i^p, h_i^q) for one root mean square (RMS) computation. Evaluation of the precision is started after RBS computes the first clock conversion coefficients, i.e. after 1000 messages have been received.

Figure 6.3 shows the cumulated distribution of the simulated precision for the LSDC and the RBS scheme. It can be seen that the precision achieved by both algorithms is comparable for no load and load one. The precision of RBS degrades in the case of additional load on one of the synchronized nodes, while LSDC is not affected. The authors of RBS report in [8] achieving microsecond precision with much less messages than were used in this experiment. There is no contradiction in this: The number of required messages is scenario dependent, therefore quick drift variations and variable load conditions require frequent synchronization. It seems that the scenarios we have studied are more severe than those evaluated in [8].

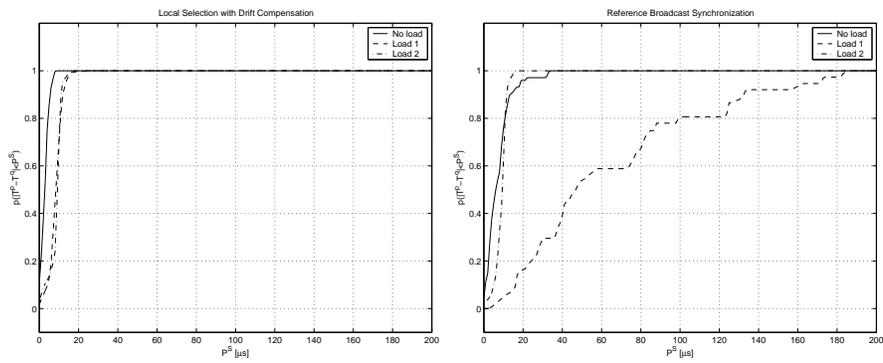


Figure 6.3: Cumulated distribution (CDF) of the precision achieved in simulation. Left: LSDC. Right: RBS scheme. The LSDC achieves low-microsecond precision under both load conditions. The precision achieved by RBS degrades in the asymmetrical load scenario.

Chapter 7

Conclusion

We found algorithms (LSA and LSDC) that achieve good precision and low jitter and that are safe and live. Most known algorithms are either not safe [6, 3, 13, 4] or not live [12]. The new algorithms scale well because they use only uni-directional communication and *can benefit* from broadcast-capable networks. The algorithms, however, *do not require* broadcast.

Experimental results show that LSDC can achieve jitter and precision on the order of $10\mu\text{s}$ on a standard wireless LAN in ad-hoc mode and Linux PCs. These parameters match with the requirements of high-quality audio distribution and the capabilities of commonly available technology (WLAN, Linux PCs). The results are comparable to those of the RBS scheme [8]. Our algorithms seem to be more robust than RBS in regard to highly variable load conditions.

The synchronization scenario described in this paper can be considered as a building block for more complex situations. For example, one may have several synchronization sources or one may use round-trip delays in order to estimate d_{\min} for an accurate synchronization. Then the results described in the present paper can be integrated into schemes to perform synchronization in large-scale wireless and ad-hoc networks and is ideally suited to be part of a service infrastructure that supports localization in time and space.

Bibliography

- [1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks. *Computer Networks*, 38(4):393–422, March 2002.
- [2] Gianluigi Alari and Augusto Ciuffoletti. Improving the probabilistic clock synchronization algorithm. In *Proceedings of the 18th Euromicro Conference*, 1992.
- [3] Gianluigi Alari and Augusto Ciuffoletti. Implementing a probabilistic clock synchronization algorithm. *Real Time Systems*, 13(1):25–46, 1997.
- [4] K. Arvind. Probabilistic clock synchronization in distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):474–487, May 1994.
- [5] 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.
- [6] Flaviu Cristian. Probabilistic clock synchronization. *Journal of Distributed Computing*, 3:146–158, 1989.
- [7] J. Elson and K. Roemer. 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.
- [8] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine grained network time synchronization using reference broadcasts. Technical Report 020008, Laboratory for Embedded Collaborative Systems LECS, UCLA, May 2002.
- [9] Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah Estrin. Locating tiny sensors in time and space: A case study. In *International Conference on Computer Design ICCD*, September 2002.
- [10] Joseph Y. Halpern, Nimrod Megiddo, and Ashfaq A. Munshi. Optimal precision in the presence of uncertainty. *Journal of Complexity*, 1(2):170–196, 1985.
- [11] J. Hill and D. Culler. A wireless embedded sensor architecture for system-level optimization. Technical Report, University California Berkeley, 2001.
- [12] Leslie Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.
- [13] Cheng Liao, Margaret Martonosi, and Douglas W. Clark. Experience with an adaptive globally-synchronizing clock algorithm. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 106–114, 1999.

- [14] Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2/3):190–204, August/September 1984.
- [15] David Mills. Network time protocol (version 3), specification implementation and analysis. IETF-RFC1305, March 1992.
- [16] David L. Mills. Internet time synchronization: The network time protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.
- [17] M. Mock, R. Frings, E. Nett, and S. Trikaliotis. Clock synchronization in wireless local area networks. In *Proceedings of the 12th Euromicro Conference on Real Time Systems*, pages 183–189, June 2000.
- [18] Rafail Ostrovsky and Boaz Patt-Shamir. Optimal and efficient clock synchronization under drifting clocks. In *Symposium on Principles of Distributed Computing*, pages 3–12, 1999.
- [19] Boaz Patt-Shamir and Sergio Rajsbaum. A theory of clock synchronization. In *Proceeding of the 26th Annual ACM Symposium on Theory of Computing, Montreal, Canada*, pages 810–819, May 1994.
- [20] K. Roemer. Time synchronization in ad hoc networks. In *ACM Symposium on Mobile Ad Hoc Networking and Computing*, October 2001.
- [21] P. Verissimo and L. Rodrigues. Cesiumspray: a precise and accurate global time service for large-scale systems. *Real-Time Systems*, 3(12):243–294, 1997.
- [22] K. Yao, R. Hudson, C. Reed, D. Chen, and F. Lorenzelli. Blind beamforming on a randomly distributed sensor array system. *IEEE Journal of Selected Areas in Communications*, 10(8):1555–1567, October 1998.