Trace-Based Evaluation of Clock Synchronization Algorithms for Wireless Loudspeakers

Philipp Blum* Lothar Thiele
Computer Engineering and Communications Laboratory (TIK)
Swiss Federal Institute of Technology Zurich (ETHZ)
e-mail: blum|thiele@tik.ee.ethz.ch

Abstract

We present an evaluation strategy for clock synchronization algorithms. It is based on a combination of measured traces, which provide for realistic performance estimation, and of simulation, which guarantees repeatability. The evaluation strategy includes parameter-optimization to allow for a fair comparison of algorithms; a general-purpose evolutionary optimizer is used for this purpose. The strategy is applied in a case study, evaluating the performance of four clock synchronization algorithms in the wireless loudspeakers application. We find that the phase-locked loop algorithm, as well as the linear-regression and the gradient algorithm achieve sufficient synchronization in a lightly loaded network. Only the local selection algorithm is able to maintain sufficient synchronization under heavy network load, as generated for example by concurrent audio or video streaming.

1 Introduction

In recent years, many new consumer-electronics 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 example in home cinema systems that connect a central source with up to eight loudspeakers distributed all over the living room. In comparison to wired networks, the message delay in wireless networks is highly variable. This is problematic for the loudspeakers application, since a few microseconds offset between correlated audio channels, i.e., stereo or surround sound channels, can cause undesirable psychoacoustic effects, as illustrated in Fig. 1. Proper synchronization of the loudspeaker nodes and the audio source is thus essential and can be achieved by clock synchronization algorithms (CSAs). Current 802.11 implementations include such mechanisms for network related purposes (e.g., centralized and distributed TSF or Last-Symbol-On-Air [6]). But these solutions do not make the synchronized time available to applications, thus the offset between correlated audio channels cannot be corrected. In this paper, we examine the questions i) whether sufficiently accurate synchronization can be achieved at the application layer, using current, off-the-shelf 802.11b hardware, and ii) which kind of CSA can do this job best.

Evaluation of CSAs based on in-system measurements provides realistic results, but has limited repeatability, which makes the comparison of different CSAs dubious. Trace-based simulation of CSAs resolves this problem. Still we are interested in realistic results.

*The author was supported by the Swiss Commission for Technology and Innovation CTI/KTI, Grant 4957.2.

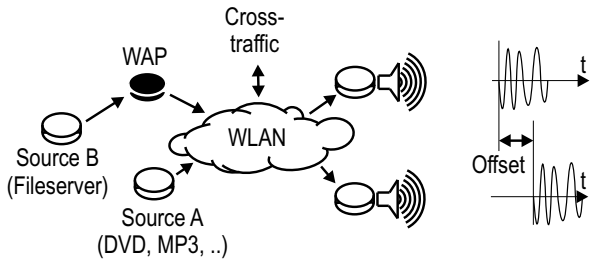

Figure 1: Case study *Wireless Loudspeakers*: Two or more loudspeakers are connected over a wireless network to an audio source. The source is either directly connected to the wireless network (A) or via a wireless access point (B). A temporal offset between the audio channels causes undesirable psychoacoustic effects: If the offset is more than $10\mathrm{ms}$, the channels are perceived as separate sources. An offset of more than $1\mathrm{ms}$ modifies the sound's characteristics (timbre). The directional perception of the source is influenced by an offset of more than $100\mu\mathrm{s}$. Rapid offset fluctuations in the range of $10\mu\mathrm{s}$ cause noise.

Therefore the traces are obtained by measurements in a real system. As the performance of a CSA is strongly affected by the choice of its parameters, we propose to regard parameter optimization as an integral step of the evaluation procedure. We have used a standard evolutionary optimizer for this purpose. For every CSA, the optimizer evaluates the same number of potential parameter sets, which provides a certain fairness regarding the effort of finding good parameters. While most presentations of new CSAs are accompanied by some selective performance evaluations, this paper is to our best knowledge the first realistic, reproducible and fair comparison of the performance of CSAs.

Section 2 describes the evaluation strategy and Section 3 presents the results of the wireless loudspeakers case study. We show that the required synchronization lies at the limit of the achievable: While all examined CSAs succeed in an unloaded network, only one of the examined CSAs can maintain sufficient synchronization under heavy cross traffic.

2 Evaluation Strategy

In this section, we describe the evaluation strategy for CSAs. Figure 2 illustrates the various steps and can serve as a quick reference for symbols and terms. First we explain the use of message-delay traces for the simulation of CSAs. Then we introduce performance metrics and explain how multicriterion optimization with evolutionary algorithms can be used to find good parameter sets for CSAs.

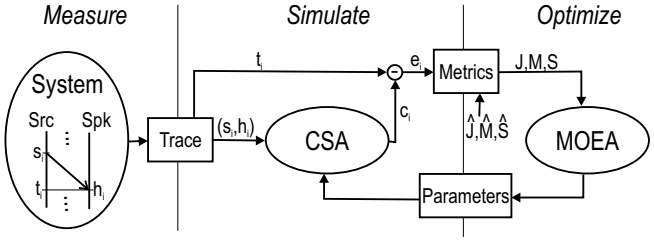

Figure 2: Evaluation Strategy for CSAs. *Measure:* Send and receive time stamps s_i and h_i are stored together with reference time t_i. *Simulate:* CSAs are simulated on the trace, producing estimates c_i. With the reference time, the synchronization error e_i can be determined. *Optimize:* The parameters of the CSA are optimized using a multicriterion-optimization evolutionary algorithm (MOEA).

2.1 System Traces

Our evaluation of CSAs is based on traces of time-stamp messages produced by a source node and received by the loudspeaker nodes.

Def. 1 (Trace) *Let s_i be the send time of the i-th time-stamp message according to the sender's clock. Let h_i be the receive time of this message according to the receiver's clock and let t_i be the corresponding reference time, that is the receive time according to the sender's clock. Then a trace with length I is the sequence of triples (s_i, h_i, t_i) with $i \in [1..I]$.*

Such a trace contains all information relevant to synchronization: As s_i is the send time and t_i is the receive time according to the source node, the message delay is $d_i = t_i - s_i$. Since h_i and t_i refer to the same instant in time, the drift of the loudspeaker node's local clock is $(h_i - h_{i-1})/(t_i - t_{i-1}) - 1$. If messages are produced periodically, we use Δt to denote the length of the time interval between consecutive messages.

2.2 Clock Synchronization Algorithms (CSAs)

In this study, we consider CSAs that match the following definition.

Def. 2 (CSA) *CSAs are local algorithms on the loudspeaker nodes and compute estimates c_i of reference time t_i. An estimate is computed upon arrival of a message from the source node containing the time stamp s_i, which is received by the loudspeaker node at local time h_i. For the computation of the estimate c_i, the algorithm can use the information of all previously received messages, that is s_j and h_j with $j \in [1..i]$.*

The concrete CSAs that are evaluated in the case study are described in Sec. 3. Note that the CSAs do not know the t_is and therefore are ignorant about message delays and clock drifts.

According to the above definition, the loudspeaker node is a pure sink of information. Many algorithms [9],[8] require also the ability to send messages from the receiver back to the sender of time-stamp messages. The presented evaluation strategy can of course also be applied to these type of CSAs. For example for CSAs that do round-trip delay measurements, the trace would consist of tuples (q_i, s_i, h_i, t_i), where the time stamp q_i is the send time of the query message produced by the receiver to trigger a time stamp message from the sender.

The advantage of the simpler unidirectional model for the wireless loudspeakers application is that the amount of time-stamp messages in the network is independent of the number of receivers and thus even a short message interval Δt causes no significant network load.

2.3 Metrics

The synchronization error $e_i = c_i - t_i$ is determined using the receive times according to the source's clock t_i from the recorded trace. The accuracy A, the peak jitter J and maximal time interval error M are defined on a suffix of the error sequence, ignoring an initial phase of real-time length $\hat{S}$, which is the *target setup-time*. On the other hand, we use a-priori defined target values $\hat{A}$ for the accuracy, $\hat{J}$ for the peak jitter and $\hat{M}$ for the MTIE to compute the real setup time S. The target values $\hat{S}$, $\hat{A}$, $\hat{J}$ and $\hat{M}$ serve the purpose of modeling the requirements of the wireless loudspeakers application.

Accuracy: The accuracy A is the maximal absolute value in the error sequence $(e_i), i \in [\lceil \hat{S}/\Delta t \rceil..I]$.

$$A = \max_i \{|e_i|\}$$

Peak Jitter: The peak jitter J is the maximal difference between two samples of the error sequence $(e_i), i \in [\lceil \hat{S}/\Delta t \rceil..I]$.

$$J = \max_i \{e_i\} - \min_i \{e_i\}$$

Maximum Time-Interval Error (MTIE): The maximum time interval error M is the maximal difference between two samples of the error sequence $(e_i), i \in [\lceil \hat{S}/\Delta t \rceil..I]$, within an interval of real time length τ.

$$M = \max_i \{ \max_{i \le j \le i + \lceil \tau/\Delta t \rceil} \{e_j\} - \min_{i \le j \le i + \lceil \tau/\Delta t \rceil} \{e_j\} \}$$

Setup Time: The setup time is the smallest value S, such that on the error sequence $(e_i), i \in [\lceil S/\Delta t \rceil..I]$, where Δt is the message interval, the accuracy A is below the target value $\hat{A}$, the peak jitter J is below the target value $\hat{J}$ and the MTIE M is below the target value $\hat{M}$.

The accuracy and the peak jitter are standard metrics in clock synchronization and for the wireless loudspeakers application describe the susceptibility to pan, timbre modification and source separation effects, see Fig. 1. The MTIE is a commonly used metric in the networking community (our definition follows [13]) and describes the susceptibility to noise.

According to the above definitions, meeting the target setup time ($S \le \hat{S}$) implies that also the accuracy, peak jitter and MTIE target values are met. Missing the target setup time ($S > \hat{S}$) implies that at least one of the other target values is missed too. For the sake of clarity we define the performance penalty P as a combination of the other metrics:

Performance Penalty: Performance penalty $P = 1$ or $P < 1$ means that all target values are achieved. $P > 1$ describes the factor by which peak jitter or MTIE fall short of the target values at the target setup time. Formally, P is defined as

$$P = \begin{cases} S/\hat{S} & \text{if } S \le \hat{S} \\ \max\left\{A/\hat{A}, J/\hat{J}, M/\hat{M}\right\} & \text{otherwise} \end{cases}$$

Mode	*Source*	*Load*	*Length*	*Interval*	$d_{\min}$	d_{median}	d_{avg}	$d_{\max}$
Ad-hoc	A (wireless)	-	$50'000$	20ms	811μs	827μs	837μs	53533μs
		MP3, 128kb/s	$50'000$	20ms	803μs	1184μs	1379μs	5686μs
		DivX, $3Mb/s$	$50'000$	20ms	809μs	1458μs	1850μs	17559μs
Infr.		-	$50'000$	20ms	1638μs	1858μs	1888μs	45737μs
		MP3, 128kb/s	$50'000$	20ms	1636μs	1875μs	2017μs	50683μs
		DivX, 3Mb/s	$50'000$	20ms	1637μs	2014μs	2605μs	82349μs
	B (wired)	-	$50'000$	20ms	1207μs	1420μs	1459μs	87800μs
		MP3, 128kb/s	$50'000$	20ms	1216μs	1472μs	1770μs	75181μs
		DivX, 3Mb/s	$50'000$	20ms	1192μs	1592μs	1914μs	50812μs

Table 1: Characteristics of the delay and drift traces used for the parameter optimization of the CSAs. With source A (see Fig. 1), IEEE 802.11 ad-hoc and infrastructure mode have been evaluated; with source B, only the infrastructure mode is possible. In all three scenarios, three different types of cross traffic (Load) have been evaluated: i) No load, ii) fixed rate MP3 stream and iii) variable rate 3Mb/s video stream.

2.4 Multicriterion Optimization with Evolutionary Algorithms

The performance of a CSA strongly depends on the setting of the CSA's parameters. Optimization of these parameters is therefore necessary. In our study, we use the SPEA2 evolutionary algorithm [14] to solve this problem. We use the implementation of the general-purpose optimization tool PISA [3]. While the actual optimizer is completely problem-independent, some elements of the evolutionary algorithm have to be customized. Evolutionary algorithms work on generations of individuals, which represent possible solutions to the optimization problem.

Individuals: An individual is a sequence $(p_i),\ i \leq \kappa$ of a CSA's parameters, where κ is the number of parameters. Most parameters are real numbers, some are integers. For every individual, the CSA is simulated and the performance metrics defined in Sec. 2.3 are evaluated.

New generations are created iteratively by removing individuals with bad performance metrics (survival of the fittest) and by creating new individuals that are similar to existing individuals with a good performance (recombination and mutation).

Recombination: Two parent individuals A and B are recombined to create two new individuals C and D by randomly selecting an integer number $i \leq \kappa$ 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 random number $i \leq \kappa$ selects one parameter, which is then modified according to $p_i := f \cdot p_i$ with a random number $f \in [0.5, 1.5]$.

The described optimization procedure provides fairness regarding the effort spent on finding good parameters for the various CSAs under consideration, as for every CSA, the same number of parameter sets are evaluated and the optimizer is general-purpose, i.e., it is not tuned to meet the needs of a particular CSA.

3 Experimental Study

In this section, we apply the evaluation strategy presented in the last section to the wireless loudspeakers application. We use a target setup time of $\hat{S} = 10s$, the target accuracy is $\hat{A} = 1\text{ms}$ to avoid timbre-distortion and source-separation effects, and the target peak jitter is $\hat{J} = 100\mu s$ to avoid pan. The target MTIE is $\hat{M} = 10\mu$s in an interval of $\tau = 10s$ to avoid noise.

In the following, we evaluate the synchronization of a *single* loudspeaker node relative to a source node, while the loudspeakers application requires synchronization among two or even more loudspeakers. The justification of this simplification is the following: (i) We evaluate CSAs that do not generate any traffic, thus the source can broadcast time-stamp messages to any number of loudspeaker nodes. (ii) The performance metrics of one loudspeaker node relative to another loudspeaker node are at most twice as large as those relative to the source's clock. This factor does not change the conclusions drawn from the results presented in this section.

3.1 Traces

In a system consisting of standard Linux PCs connected by an IEEE 802.11b network, we recorded traces with periodic send times $s_i = s_0 + i\Delta t$, with a send interval $\Delta t = 20\text{ms}$ and length $I = 50000$. The traces were recorded in a system consisting of four Pentium II Linux PCs, connected by an IEEE 802.11b network in ad-hoc and infrastructure mode. Table 1 gives details about the load conditions and statistical properties of these traces. We used the time-stamp counter register (TSC) of the Pentium CPU for the local time of the nodes. As the TSC is incremented at each processor cycle, it has a resolution in the nanosecond range. The time-stamping facility was implemented as a loadable kernel module to reduce OS-latencies that otherwise would have caused additional delay variability. The reference times t_i were measured using a simple cable connection between all nodes' parallel ports to trigger simultaneous interrupts on all nodes. A detailed description of the measurements is given in [4], similar procedures are described in [9] and [7]. The accuracy of these measurements is better than $\pm 5\mu$s.

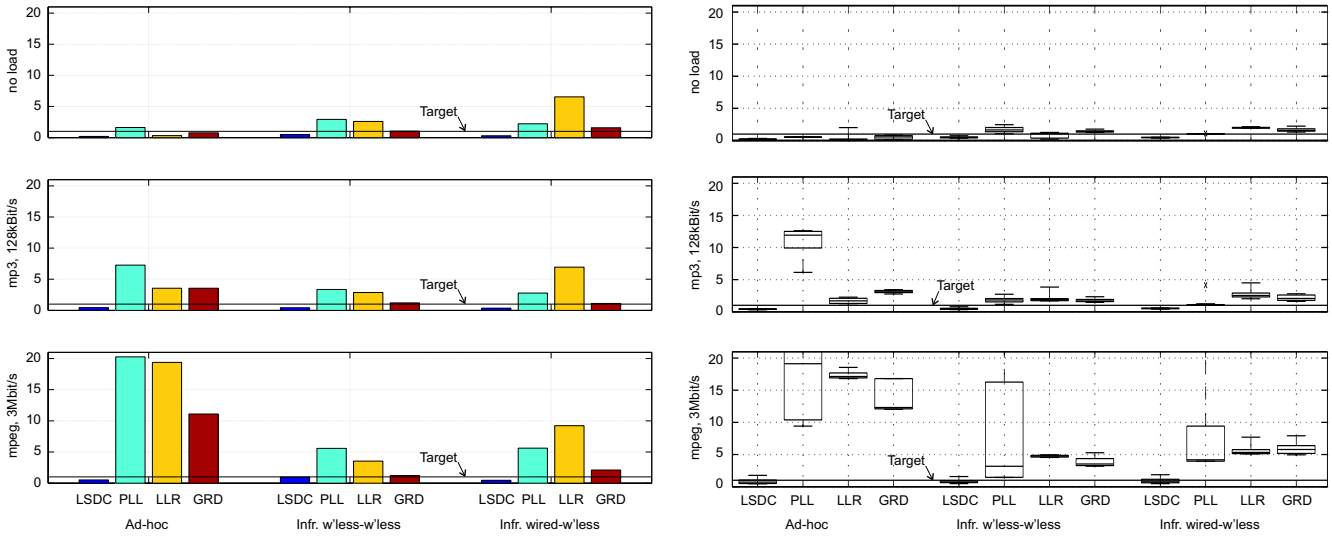

Figure 3: In (a), the performance penalty P of the Local Selection (LSDC), the Phase-Locked Loop (PLL), the Linear Regression (LLR) and the Gradient (GRD) algorithms in the three operation modes. The top row shows the results in an network without load, the two rows below show the effect of increasing network load. For every scenario and load type, the parameterization was optimized individually. In (b), box plots of the performance penalty P achieved on ten traces per scenario and load type. In contrast to (a), a single parameterization is used for all scenarios and load types. The median penalty and the computation overhead of the algorithms used in this figure are summarized in Tab. 2.

3.2 Algorithms

We compare the performance of four classes of CSAs: (i) The Local Selection (LSDC) algorithm that has been presented in [5]. (ii) A Phase-Locked Loop (PLL) algorithm, used for example in the NTP [8][1]. (iii) A Linear Regression (LLR) algorithm. Such algorithms have been presented and analyzed in [2] and [10]; we use the implementation of [11]. An LLR algorithm is also the core of the Reference Broadcast Synchronization (RBS) algorithm [7]. (iv) The Gradient (GRD) algorithm, which is quite similar to the LLR, but requires less memory. Pseudo-code descriptions of all algorithms and their parameters can be found in the appendix.

3.3 Experimental Setup

An 802.11 WLAN can be operated in different modes, depending whether an access point is present or not. We compare the performance of the CSAs in (i) a network in ad-hoc mode and the source node is a wireless node, (ii) a network in infrastructure mode and source node is a wireless node, (iii) a network in infrastructure mode and the source node is part of a wired network that is connected to the wireless network via an access point. In addition, we vary the network load. An overview of the different scenarios and some statistical properties of the traces recorded in these scenarios is shown in Tab. 1.

The evaluation of a CSA consists of evolving 40 individuals over 100 generations. From the last generation, the individual with the smallest performance penalty is selected as the best parameter set and the corresponding performance metrics are then compared with those achieved by other CSAs.

Figure 3 (a) shows the achieved performance penalty P in the different scenarios and network-load types. The algorithms were optimized for every scenario individually. Figure 3 (b) shows the achieved performance penalty P of one single algorithm optimized for all scenarios concurrently. For every scenario and load type, ten traces were evaluated and the resulting Ps are displayed using box plots[2], the median values are summarized in Tab. 2.

3.4 Discussion

Consider the performance penalty of the individually optimized algorithms, shown in Fig. 3 (a). For all algorithms and scenarios, P increases with increasing network load. The negative impact of the network load is more accentuated in the ad-hoc scenario than in the infrastructure scenarios. The LSDC algorithm undercuts all target metrics ($P < 1$) in all scenarios and load types. The GRD algorithm achieves an almost sufficient P with no load and in the infrastructure scenarios. Both PLL and LLR clearly fail in all scenarios with network load. Now we compare these results with those from Fig. 3 (b). The penalties here are slightly worse because the CSAs' parameters are not optimized for every scenario individually. The PLL algorithm seems to be least robust of the examined CSAs, in the sense that its performance penalty under heavy network load is much more variable than that of the other algorithms.

Only the LSDC algorithm achieves all target metrics in all scenarios and with all load types. For the other algorithms, it is difficult to achieve reasonable peak-jitter values J under heavy network

[1]The actual NTP incorporates many additional features and is based on round-trip messages, thus it can not be used here directly.

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

Algorithm	*Ad-hoc*			*Infr., w'less-w'less*			*Infr., wired-w'less*			*Operations*				*Memory*
	-	128kb/s	3Mb/s	-	128kb/s	3Mb/s	-	128kb/s	3Mb/s	n^+	$n^<$	n^*	$n^/$	
LSDC	0.20	0.38	0.72	0.62	0.62	0.89	0.46	0.52	0.76	13	6	5	3	43
PLL	0.52	11.92	19.12	1.83	1.85	4.33	1.07	1.05	4.19	8	3	5	0	7
LLR	0.16	1.69	17.14	1.02	1.98	4.90	1.96	2.54	5.33	15	4	10	2	43996
GRD	0.56	3.20	12.30	1.20	1.77	4.15	1.59	2.07	5.78	13	4	4	5	8

Table 2: Median performance penalty P and resource requirements of the algorithms used for the results from Fig. 3 (b). The table shows the number of operations sorted by the type of operation (n^+: additions and subtractions, $n^<$: comparisons, n^*: multiplications, $n^/$: divisions and modulo operations) and the number of time stamps that are stored in memory. In our implementation, time stamps are 8 bytes long.

load. This study thus confirms the advantage of the LSDC algorithm claimed in [5]. It is explained by the fact that in contrast to all other algorithms, the LSDC 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. 1. Figure 3 also indicates that the ad-hoc scenario with high load is most difficult for the PLL, LLR and GRD algorithms. This matches with the variability of the median delay under network load, shown in Tab. 1. While d_{median} in the ad-hoc scenario varies by more than 600μs, it varies by less than 200μs in the infrastructure scenarios.

In Tab. 2, the cost of the algorithms is shown. The PLL and the GRD are cheapest, the LSDC is more expensive, particularly regarding memory requirements. The LLR uses much more memory than the others.

4 Conclusion

We have presented the first realistic, reproducible and fair performance evaluation strategy for clock synchronization algorithms. The proposed procedure produces *realistic* estimations of the achievable performance since it is based on *measured* system traces. The results are *reproducible* because they are obtained with simulation on the recorded traces. A comparison of various algorithms is *fair*, because the algorithms are evaluated on the same system traces and because the proposed procedure includes automated parameter optimization. In previous work, evaluation of CSAs was based on theoretical properties rather than quantitative metrics [1, 12], or was limited to selective, non-reproducible measurements [7, 9].

The proposed evaluation strategy has been applied to the application of wireless speakers. This experimental study has shown that the rigorous requirements of this application can be achieved even under heavy cross traffic with the LSDC algorithm, which selectively synchronizes to time-stamp messages with a delay that is close to the minimal delay. Other algorithms that synchronize to average-delay messages fail to maintain sufficient synchronization under cross traffic.

The proposed evaluation strategy can easily be applied to other applications. We have proposed a set of metrics which allow to accurately model any application's synchronization requirements. Traces can be obtained from any target platform, which allows to account for the peculiarities of a given platform already during evaluation and optimization of various synchronization algorithms.

5 Acknowledgments

Thanks to Christian Hitz and Stefan Schuler who have implemented an early version of the parameter optimization step and to Georg Dickmann from BridgeCo AG for his comments on the requirements of loudspeakers applications.

References

[1] Emmanuelle Anceaume and Isabelle Puaut, *Performance evaluation of clock synchronization algorithms*, Tech. Report 3526, IRISA, Rennes, France, October 1998.

[2] K. Arvind, *Probabilistic clock synchronization in distributed systems*, IEEE Transactions on Parallel and Distributed Systems **5** (1994), no. 5, 474–487.

[3] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler, *Pisa – a platform and programming language independent interface for search algorithms*, Evolutionary Multi-Criterion Optimization (EMO 2003), LNCS, vol. 2632, 2003.

[4] Philipp Blum and Georg Dickmann, *Precise delay measurements in wired and wireless local area networks*, Tech. Report 175, Computer Enigneering and Networks Lab., ETH, Zurich, Switzerland, July 2003.

[5] Philipp Blum and Lothar Thiele, *Clock synchronization using packet streams*, International Symposium on Distributed Computing (Short paper), October 2002.

[6] Javier del Prado, Sai Shankar, and Sunghyun Choi, *Multimedia clock synchronization over IEEE 802.11 WLAN*, New York Metropolitan Area Networking Workshop (NYMAN03), September 2003.

[7] Jeremy Elson, Lewis Girod, and Deborah Estrin, *Fine-grained network time synchronization using reference broadcasts*, Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002), December 2002.

[8] David L. Mills, *Internet time synchronization: The network time protocol*, IEEE Transactions on Communications **39** (1991), no. 10, 1482–1493.

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

[10] Raffaele Noro, *Synchronization over packet-switched networks: Theory and applications*, Ph.D. thesis, EPFL, Lausanne, Switzerland, 2000.

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

[12] B. Simons, J. Lundelius-Welch, and N. Lynch, *An overview of clock synchronization*, Fault-Tolerant Distributed Computing, Springer Lecture Notes on Computer Science 448 (A. Spector B. Simons, ed.), 1990, pp. 84–96.

[13] International Telecommunication Union, *Definitions and terminology for synchronization networks*, ITU-T Recommendation G.810, 1996.

[14] Eckart Zitzler, Marco Laumanns, and Lothar Thiele, *SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization*, Proceedings of the EUROGEN2001 Conference, Athens, Greece, September 19-21, 2001, 2002, pp. 95–100.

A Algorithms in Pseudocode

The following algorithms are CSAs according to Def. 2. The output is a function C_i that maps any local-clock time $H(t)$ with $t \geq t_i$ to a synchronized time $C_i(H(t))$. The synchronized time immediately after execution of the CSA is $c_i = C_i(H(t_i)) = C_i(h_i)$. The input includes all pairs (h_j, s_j) with $j \leq i$ and quantites derived from these values, e.g., the function C_{i-1}.

Algorithm 1 Algorithm LSDC, approx. following [5]

Parameters: Initial phase ι
Drift compensation factor $(\alpha_{\max}, \alpha_{min}, \alpha_\mu)$
Leakage factor $(\lambda_{\max}, \lambda_{min}, \lambda_\mu)$
State: Drift estimation $r := 0$
Drift compensation factor $\alpha := \alpha_{\max}$
Leakage factor $\lambda := \lambda_{\max}$

// start new clock with received time stamp
$c_i := s_i$
if $i > \iota$ **then**
  // slow down clock
  $r := r + \lambda(h_i - h_{i-1})$
  if $s_i > C_{i-1}(h_i)$ **then**
    // speed up clock
    $r := r - \alpha(s_i - C_{i-1}(h_i))$
    // decrease drift compensation parameters
    $\lambda := (1 - \lambda_\mu)\lambda + \lambda_\mu \lambda_{\min}$
    $\alpha := (1 - \alpha_\mu)\alpha + \alpha_\mu \alpha_{\min}$
  else
    // ignore current input s_i
    $c_i := C_{i-1}(h_i)$
  end if
end if
$C_i(H(t)) := c_i + \frac{H(t)-h_i}{1+r+\lambda(H(t)-h_i)}$

Algorithm 2 Algorithm PLL, following [10], Section 3.2

Parameters: Integral gain κ_{I}
Proportional gain κ_{P}
Maximal input signal $\theta_{\max}$
State: Integrator sum $S_{\mathrm{I}} := 0$
Temporary variables: Input signal θ

if $i > 1$ **then**
  $\theta := s_i - C_{i-1}(h_i)$

  // limit input
  $\theta := \max(-\theta_{\max}, \min(\theta_{\max}, \theta))$

  // update integrator sum
  $S_{\mathrm{I}} := S_{\mathrm{I}} + \kappa_{\mathrm{I}}(h_i - h_{i-1})\theta$

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

Algorithm 3 Algorithm LLR, following [11], Section 15.2

Parameters: Averaging-window size κ
State: $S_{\mathrm{loc}} := S_{\mathrm{ref}} := S_{\mathrm{loc2}} := S_{\mathrm{locref}} := 0$

// insert new sample
$S_{\mathrm{loc}} := S_{\mathrm{loc}} + h_i$
$S_{\mathrm{ref}} := S_{\mathrm{ref}} + s_i$
$S_{\mathrm{loc2}} := S_{\mathrm{loc2}} + h_i^2$
$S_{\mathrm{locref}} := S_{\mathrm{locref}} + h_i * s_i$

if $i > \kappa$ **then**
  // remove oldest sample
  $S_{\mathrm{loc}} := S_{\mathrm{loc}} - h_{i-\kappa}$
  $S_{\mathrm{ref}} := S_{\mathrm{ref}} - s_{i-\kappa}$
  $S_{\mathrm{loc2}} := S_{\mathrm{loc2}} - h_{i-\kappa}^2$
  $S_{\mathrm{locref}} := S_{\mathrm{locref}} - h_{i-\kappa} * s_{i-\kappa}$
end if

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

Algorithm 4 Algorithm GRD

Parameters: Averaging-window size κ, Initial phase ι
State: $S_{\mathrm{loc}} := S_{\mathrm{ref}} := D_{\mathrm{loc}} := D_{\mathrm{ref}} := 0$

// update predictor
$S_{\mathrm{loc}} := S_{\mathrm{loc}} + h_i$
$S_{\mathrm{ref}} := S_{\mathrm{ref}} + s_i$
if $i > \iota$ **then**
  $D_{\mathrm{loc}} := (\kappa - 1)/\kappa * D_{\mathrm{loc}} - 1/\kappa * S_{\mathrm{loc}}/(i-1)$
  $D_{\mathrm{ref}} := (\kappa - 1)/\kappa * D_{\mathrm{ref}} - 1/\kappa * S_{\mathrm{ref}}/(i-1)$
end if

// predict
if $D_{loc} > 0$ **then**
  $C_i(H(t)) := S_{\mathrm{ref}}/i + (H(t) - S_{\mathrm{loc}}/i) * D_{\mathrm{loc}}/D_{\mathrm{ref}}$
else
  $C_i(H(t)) := S_{\mathrm{ref}}/i + (H(t) - S_{\mathrm{loc}}/i)$
end if