On the Time Synchronization of Distributed Log Files in Networks with Local Broadcast Media

Björn Scheuermann, Wolfgang Kiess, Magnus Roos, Florian Jarre, and Martin Mauve

Abstract— **Real-world experiments in computer networks typically result in a set of log files, one for each system involved in the experiment. Each log file contains event timestamps based on the local clock of the respective system. These clocks are not perfectly accurate, and deviate from each other. For a thorough analysis, however, a common time basis is necessary. In this paper, we tackle the fundamental problem of creating such a common time base for experiments in networks with local broadcast media, where transmissions can be received by more than one node. We show how clock deviations and event times can be estimated with very high accuracy, without introducing any additional traffic in the network. The proposed method is applied after the experiment is completed, using just the set of local log files as its input. It leads to a large linear program with a very specific structure. We exploit the structure to solve the synchronization problem quickly and efficiently, and present an implementation of a specialized solver. Furthermore, we give analytical and numerical evaluation results and present real-world experiments, all underlining the performance and accuracy of the method.**

I. Introduction

A fundamental problem with interpreting results from real-world experiments in computer networks is that each system uses its own local clock to timestamp events. These clocks do not run perfectly synchronously, they can deviate. At the end of the experiment, the result is a set of log files where the timestamps are based on the clocks of the individual systems. This is generally insufficient for the investigation of timing issues, the correlation of events, and the visualization of the experiment. It is thus highly desirable to obtain an event log where all timestamps refer to a single reference clock instead of multiple local clocks.

The obvious attempt to reach this goal is to synchronize the clocks of the involved systems with a time synchronization protocol like NTP [1], [2]. Unfortunately, there are two key reasons why this approach may be inappropriate. First, running a time synchronization protocol will cause additional network traffic and may thus interfere with the experiment. Second, even if the clocks were perfectly synchronized, it takes some system dependent (and potentially non-deterministic) time from the occurrence of an event until it is actually time-stamped and recorded. We call this the *timestamping delay*. The drawback of additional network traffic can be eliminated by providing each system with a very accurate clock, e. g., controlled by a GPS receiver. Aside from the fact that this is quite expensive, this approach does not solve the problem of the timestamping delay. While it may also be possible to use customized hard- and software to bound the timestamping delay, such a solution cannot be employed for the off-the-shelf systems often used in network experiments.

In order to avoid these problems we propose to correct the timestamps of the individual log files after an experiment is completed instead of synchronizing clocks during the experiment. To do so, we exploit a specific characteristic of networks with local broadcast media: a transmission is often received by multiple nodes. Upon recording this transmission, each node uses its local clock to provide a timestamp for the same physical event. Such shared events can be used as *anchor points* that relate the different clocks to each other. The combination of multiple anchor points allows for a very good estimation of this relation and finally for an accurate post-experiment synchronization of the log files.

In essence we employ a model for the clocks and the timestamping delays. We then use the anchor points to estimate the parameters of this model and thus the clock deviations. This results in estimates for the timestamps of all events on a common time basis. A maximum likelihood estimator is used for this purpose. It leads to a large linear program with a very specific structure. We exploit this structure to solve the linear problem efficiently in spite of its huge size. The solution then yields a synchronized log file where all entries are recorded with a common time basis.

Analytical and numerical results show that the solution converges quickly to a good estimate for increasing input data sizes, and that it is robust if the assumptions made for its derivation are not perfectly fulfilled. Thus, in practice, a very reasonable amount of log data is typically sufficient to identify and eliminate clock deviations.

Our approach is applicable to all networks with local broadcast characteristics. It just requires that the clocks of any two nodes in the network can be—directly or indirectly—set into relation by anchor points. In particular, this includes experiments in wireless ad-hoc-, sensor- and mesh-networks, as well as local area networks with multiple stations in each collision domain and satellite networks.

The key contribution of this work is an algorithm to synchronize the distributed log files in networks with local broadcast media. In detail this contribution includes: (1) a maximum likelihood estimator for the clock parameters and global event times, based on models for the local clocks and the timestamping delays, (2) the expression of the maximum likelihood estimator as the solution of a linear program, (3) an efficient way to solve this specific type of linear program, (4) a publicly available implementation, (5) analytical error bounds and a consistency proof, i. e., convergence to a correct solution for increasing data set sizes, (6) a numerical evaluation of the algorithm and its implementation, and (7) an experimental evaluation of the synchronization performance.

This paper is structured as follows. In the next section, we review related work and examine its applicability to the synchronization of distributed log files. We then introduce a detailed clock model and establish the required terminology in Section III. Section IV presents our synchronization algorithm and Section V shows how the resulting optimization problem can be solved efficiently. The properties of the algorithm are analytically investigated in Section VI and are verified through a numerical evaluation in Section VII. Subsequently, we apply our algorithm to real-world experimental data in Section VIII, and analyze the results. Section IX concludes this paper with a summary.

II. Related Work

The relevant literature in the area of clock synchronization can be divided into online and offline clock synchronization protocols. The aim of *online clock synchronization* protocols, like the well-known Network Time Protocol (NTP) [1], [2], is to keep the clocks of the participating nodes synchronized while the network is up and running. In contrast to that, *offline clock synchronization* approaches correct timestamps that have been provided by unsynchronized clocks after the experiment is finished. Our own approach clearly falls into the second category.

A. Online clock synchronization

As discussed in the introduction, online approaches typically use explicit messages for clock synchronization. They are also constrained by the fact that they need to work in a distributed fashion and may consume only very limited computational resources. Moreover, online synchronization can only exploit past information, whereas offline approaches can make use of all—previous as well as later occurring—events for the time estimates. For these reasons, online synchronization protocols are not an optimal solution for the synchronization of distributed log files. Nevertheless, some of them use the idea of events that are observed by multiple systems. In the following, we summarize those approaches. A broader overview of the topic, with a focus on wireless sensor networks, can be found in [3].

A number of online synchronization protocols [4]–[6] relies on the parallel reception of broadcasted packets by multiple systems. A broadcasted packet is received by all systems nearly at the same instant, the only uncertainty in timestamping such packets is the signal propagation time and the timestamping delay. To synchronize the clocks, the recipients of a given broadcast communicate to exchange their respective reception times. By comparing these reception times, two nodes are able to compare and adjust their clocks. In [6], for example, the clock skew is estimated using linear least-squares regression. A complete network can then be synchronized by synchronizing adjacent nodes pairwise along a tree structure, yielding, however, the disadvantage of accumulating the pairwise errors.

In [7], the pairwise synchronization of [6] is extended to a global one. The authors present an online synchronization approach for sensor networks that is based on a global unbiased minimum variance estimator. They first introduce a version that considers only clock offsets, and then complement it with an idea on how to deal with clock rate differences. Their approach is, however, not able to handle offsets and rate deviations conjointly, but must rely on separate estimates on different time scales. This is feasible and appropriate in the considered context of online time synchronization for continuously running sensor networks, but is not optimal for the offline synchronization of the logs of time-limited experiments. In addition to avoiding the general drawbacks of using online approaches for the synchronization of log files, our approach estimates offsets and rates in one single step, and can thus exploit all the available information to find the global optimum for both.

B. Offline clock synchronization

The first offline clock synchronization algorithm has been proposed by Duda et al. [8] for generic distributed systems. The send and receive timestamps of messages between nodes A and B are taken as coordinates of a point, the x-axis being the timestamp of A and the y-axis being the timestamp of B for the same packet. Due to the network delay, two point-clouds emerge with an empty corridor in between. Each point is either above the corridor (when sent from A to B) or below (when sent from B to A). The authors present two methods to fit a line in this corridor, thereby estimating the difference in clock speed and offset between A and B. The first method computes the separating line with linear regression, the other uses a convex hull approach. They also sketch a maximum likelihood approach but are not able to use it due to a lack of knowledge about the message delay from sender to receiver, which would be needed.

Duda's linear regression and convex hull approaches have been extended in [9]. The author corrects the timestamps using experimental knowledge about the smallest round trip delays. This knowledge is incorporated in an algorithm that selects the two best points to estimate the skew and offset between the nodes. In [10], linear programming is used to compensate for clock skew that influences one-way delay measurements between two nodes over the Internet. A convex hull based approach able to cope with clock resets is presented in [11].

All of the presented offline synchronization algorithms can compensate linear clock deviations between two nodes without requiring additional network traffic. In contrast to our approach, which exploits the broadcast nature of the medium, they can be used for all kinds of communication systems. However, this benefit is also their main drawback: all of them consider the comparison of send and receive timestamps. Thus, the network delay cannot be completely eliminated, as it is the case in our approach. Likewise, they cannot separate and handle the timestamping delay. Finally, while we use all the available data to compute globally consistent estimates for an arbitrary number of nodes in parallel, all these algorithms synchronize only two clocks directly. In order to synchronize more clocks, a successive synchronization of node pairs is necessary, a process in which errors can accumulate.

III. Model, Terminology, and Applicability

A. Nodes and events

In our terminology, an *event* is an incident that has been observed by one or more nodes and is recorded in their local log files. Of particular interest for us are packet reception events, since they can be observed by multiple nodes almost at the same time. Broadcast packets can generally be received by all nearby nodes, for unicast transmissions a similar effect can be achieved by logging receptions in promiscuous mode. In this mode, a node records all receptions that could be decoded by its network interface, regardless of whether the node is the intended destination of the transmission or not. We assume that parallel receptions of the same transmission can be identified as such. In the remainder of this paper, we concentrate primarily on events that have occurred in more than one node, since they can serve as anchor points for the synchronization.

We denote the set of nodes participating in an experiment by J and the set of events that occur during the experiment by I. Each event $i \in I$ occurs at some "true" time T_i. The same event i can be observed by multiple nodes. In this case each of the nodes records its own timestamp for the event according to its local clock, i. e., event i is recorded by some of the nodes $j \in J$ with local timestamps $t_{i,j}$.

The recorded times define a relation $R \subseteq I \times J$ in the sense that $(i, j) \in R$ if and only if event i is recorded by node j. The subset of nodes that observe a certain event $i \in I$ is denoted by R_i, i. e., $j \in R_i$ if and only if $(i, j) \in R$.

B. Clocks

In this paper we model clocks as twice differentiable functions, mapping some (virtual) global, absolute time to the view of the respective clock. This model matches those commonly used in literature related to clocks and time synchronization [2], [6], [10], and is justified since only the limited timespan of a single experiment needs to be considered. For the same reason and for the sake of simplicity we do not account for clock resets, although our approach could be extended to such a scenario.

The *true clock* C_T is a clock which is correct by definition: $\forall t : C_T(t) = t$. Our aim is to approximate this clock as closely as possible by the calculated global event timestamps.

We use the common terminology to describe the properties of clocks. The *offset* of a clock C at time t is the difference $C(t) - C_T(t)$ between C and the true clock C_T. If we use the term offset without referring to a certain point in time we refer to $C(0)$, the offset at time $t = 0$. $C'(t)$ is called the *rate* or *frequency* of C at time t. The difference between a clock's rate and the true clock's rate $C'(t) - C'_T(t) = C'(t) - 1$ is called *skew*. Finally, the second derivative $C''(t)$ is called the *drift* of C.

The clock model we use for our estimator assumes that, during the time interval of an experiment, the local clocks in the nodes can be closely approximated by a linear function. We denote the rate of a node j's local clock C_j by $r_j > 0$ and its offset at time $t = 0$ by o_j. Thus, for all times t during the experiment we have

$$C_j(t) = r_j t + o_j. \tag{1}$$

The time span over which the linearity assumption holds is related to the clock's frequency stability. It is commonly specified using the Allan deviation [12], which characterizes the rate variations over different timescales. One application area of offline synchronization are experiments with mobile ad-hoc networks (MANETs). MANET experiments described in the literature are surveyed and analyzed in [13]; there, it has been pointed out that typically the duration of an experimental run does not exceed 1000 seconds. This coincides with previous work, which shows that clock drift is typically negligible over time spans up to 1000 seconds [14].

For longer experiments, if the linearity assumptions do not hold, the accuracy of the results may deteriorate. As will become clear later, the degradation is graceful, i. e., the estimation is a good linear approximation. Note also that it is easily possible to synchronize arbitrary sub-intervals of longer logs, such that the assumption holds reasonably well within each sub-interval.

Furthermore, care must be taken that the linearity assumption is not thwarted by processes running on the nodes and manipulating the clocks. If online synchronization must be used—e. g., because it is part of the experiment—then it should record all the modifications it made to the local clock, so that the effects of these changes can be eliminated from the log files of the respective nodes prior to synchronizing them.

In practice, time in computer systems does not run continuously, but progresses in discrete steps. While the resolution of the timer-interrupt driven system clock is typically relatively coarse—in the order of milliseconds—, more fine-grained time sources are often available and used. On the x86 platform, for example, the CPU's TSC register progresses with every CPU clock cycle. Thus, its granularity is very fine. It serves for generating the timestamps, for example, when using a Linux kernel and the widespread packet tracing library libpcap. Thus, we can assume that the error introduced by the clock resolution is small in comparison to other sources of error. Our approach does not amplify such errors.

C. Timestamping delay

When sending a message, a number of different delays occur from the moment the source application generates the message until the receiver timestamps it. As our approach uses these timestamps as synchronization anchors, we are interested in the delay *differences* experienced by distinct nodes. The deterministic components are not an issue in our context: if all timestamps in a node are recorded late by some fixed time, then this is the same as if they were recorded immediately with a correspondingly increased offset. So, the fixed delay components are equivalent to an additional clock offset.

The experienced delay can be decomposed into four components according to [15]: the time needed to compose the message and to assemble the packet, the time to access the medium, the propagation delay on the medium, and finally, after the transmission arrives at the receiver, the *receive time*,

i. e., the delay for checking the message and recording the arrival timestamp. Obviously, the time until the packet leaves the sender is the same for all receivers and thus does not need to be considered.

The propagation delay depends on the distance between sender and receiver, and the propagation delay differences depend on the different distances between sender and receivers. As long as these differences are in the order of a few hundred meters, the propagation delay difference is in the order of, at most, microseconds and is therefore negligible[1]. The reason for the receive time is that timestamps are not assigned immediately upon reception. The timestamping process will typically be delayed. The delay can be decomposed further into a fixed component (which equals the minimum path delay of the processing necessary at the receiver), and an additional, variable time that occurs because the timestamping is performed by the node's CPU, which may be busy with other tasks before the event is processed. The latter we call *timestamping delay*.

Note that the delay of an event is also "measured" by the recording node's clock, and thus is scaled with the rate of this clock. An event i at "true" time T_i that is recorded by node j with timestamping delay $d_{i,j}$ thus leads to a timestamp

$$t_{i,j} = C_j(T_i + d_{i,j}) = r_j(T_i + d_{i,j}) + o_j. \tag{2}$$

The timestamping delay is, like all delays, obviously non-negative. Furthermore, it seems reasonable to assume that most timestamps are recorded with small latency and few are set after a longer time. We model the timestamping delays as exponentially distributed, pairwise independent random variables. Moreover, we assume that the exponential distributions of all delays share the same parameter λ. The latter is reasonable if the nodes participating in the experiment use comparable hard- and software for the timestamp generation, which will often be the case in a testbed.

In a real-world application our assumptions about the timestamping delay just like those about the clocks' linearity will of course not perfectly hold. In fact, depending on the hard- and software of the devices, reality might look very different. We use the mentioned assumptions for the motivation and derivation of our method. It will later become clear that the resulting approach yields good results also under non-conformant circumstances.

D. Connectivity Constraints

As our proposed approach relies on anchor points to relate the clocks, it depends on the presence of events that can serve this role. Consider the case in which there is no common event between two groups of nodes. Here, it would be impossible for anchor point-based synchronization to tell if all clocks in one of the groups are, for example, early by one hour. A common, global time basis can thus not be established, whereas it remains nevertheless possible to synchronize the clocks *within* each group.

[1]If nodes are really far apart and the propagation delay is long, then it is often the case that the distances and thus also the delays are approximately known. This applies to satellite systems, for example. In this case it is possible to eliminate the delay prior to synchronization.

Note that the availability of anchor points does *not* imply that all pairs of nodes must share common events—clocks may also be related indirectly, over intermediate nodes. It also does not necessitate the network to be "connected" in the commonly used sense. For example, if there are two almost independent groups of nodes, one single node sharing events with both groups suffices. These shared events need not occur during the same time intervals, and thus there is no need for a fully connected topology at even just one single point in time.

Hence, the existence of anchor points is not a severe constraint in practice, and anchor point-based synchronization will be possible in the vast majority of experimental setups. If this condition is not met, artificial anchor points could be generated, e. g., by broadcasting "anchor packets". Doing so during the experiment might interfere with the experiment itself, just like running an online synchronization protocol. Anchor points, however, may also be generated before and after an experimental run.

IV. Algorithm

The previous section introduced a model of the network and the timestamping delays. Now, we will formalize the problem and propose an approach for its solution via a maximum likelihood estimator (MLE). Given the recorded local timestamps, we wish to maximize the likelihood that our estimates of the true event times are correct.

Due to the exponentially distributed delays, the conditional probability density for measuring a timestamp $t_{i,j}$ for event i at node j given C_j and T_i is

$$f(t_{i,j} \mid C_j, T_i) = f(d_{i,j}) = \lambda e^{-\lambda d_{i,j}}. \tag{3}$$

Because of the independence of the delays the probability density for the whole set of measurements in our experiment can be written as

$$f((t_{i,j})_{(i,j)\in R} \mid (C_j)_{j\in J}, (T_i)_{i\in I}) = \prod_{(i,j)\in R} \lambda e^{-\lambda d_{i,j}}. \tag{4}$$

We can now express our problem as an optimization problem. Under a uniform prior, we want to find the optimal estimates $\widehat{T}_i$ of T_i for all $i \in I$, and, in parallel, the optimal estimates $\widehat{C}_j$ of C_j for all $j \in J$ such that the likelihood function L defined in the following way is maximized:

$$\begin{aligned} L &= L((\widehat{C}_j)_{j\in J}, (\widehat{T}_i)_{i\in I} \mid (t_{i,j})_{(i,j)\in R}) \\ &= f((t_{i,j})_{(i,j)\in R} \mid (\widehat{C}_j)_{j\in J}, (\widehat{T}_i)_{i\in I}). \end{aligned} \tag{5}$$

From (2) we can see that

$$\forall (i,j) \in R : d_{i,j} = \frac{t_{i,j} - o_j}{r_j} - T_i. \tag{6}$$

This relation must also hold for the estimates of T_i and C_j. Let $\widehat{r}_j$, $\widehat{o}_j$, and $\widehat{d}_{i,j}$ denote the estimates for r_j, o_j, and $d_{i,j}$, respectively. Then, in analogy to the above we have

$$\forall (i,j) \in R : \widehat{d}_{i,j} = \frac{t_{i,j} - \widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i. \tag{7}$$

Therefore, L can be expressed as

$$\begin{aligned} L &= \prod_{(i,j)\in R} \lambda e^{-\lambda \widehat{d}_{i,j}} \\ &= \prod_{(i,j)\in R} \lambda e^{-\lambda\left(\frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j}-\widehat{T}_i\right)}, \end{aligned} \tag{8}$$

eliminating the estimates $\widehat{d}_{i,j}$ for the unknown quantities $d_{i,j}$.

Since all the delays are non-negative, the maximization of L is subject to the constraints

$$\forall (i,j) \in R : \frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i \geq 0. \tag{9}$$

Now we apply a standard technique in maximum likelihood estimation: maximizing L is equivalent to maximizing $\ln L$, because $L > 0$ for all valid parametrizations.

$$\begin{aligned} \ln L &= \ln \prod_{(i,j)\in R} \lambda e^{-\lambda\left(\frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j}-\widehat{T}_i\right)} \\ &= \sum_{(i,j)\in R} \left(\ln\lambda + \ln e^{-\lambda\left(\frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j}-\widehat{T}_i\right)} \right) \\ &= |R| \ln\lambda - \sum_{(i,j)\in R} \lambda \left(\frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i \right). \end{aligned} \tag{10}$$

Optimizing this expression with regard to λ and all the $\widehat{T}_i$ and $\widehat{C}_j$ is a difficult nonlinear optimization problem. However, we are not primarily interested in the parameter λ. Fortunately it turns out that the optimal $\widehat{T}_i$ and $\widehat{C}_j$ are independent of the value of λ. Let for the moment

$$k(x) := -\frac{\ln x - |R| \ln\lambda}{\lambda}. \tag{11}$$

k is strictly monotonically decreasing for any $\lambda > 0$ and $|R|$. Thus, it is easy to see that L is maximal if and only if $k(L)$ is minimal:

$$k(L) = -\frac{\ln L - |R| \ln\lambda}{\lambda} = \sum_{(i,j)\in R} \left(\frac{t_{i,j}-\widehat{o}_j}{\widehat{r}_j} - \widehat{T}_i \right). \tag{12}$$

Therefore, instead of maximizing L, we minimize $k(L)$. We have thus eliminated the variable $\lambda > 0$. The constraints of the resulting optimization problem are still of the form (9).

From the clock model we know that the rates of the clocks are strictly positive. We exploit this fact and define

$$\bar{r}_j := \widehat{r}_j^{-1} \tag{13}$$

$$\bar{o}_j := \frac{\widehat{o}_j}{\widehat{r}_j}. \tag{14}$$

Equivalently, we have $\widehat{r}_j = \bar{r}_j^{-1}$ and $\widehat{o}_j = \bar{o}_j \widehat{r}_j = \frac{\bar{o}_j}{\bar{r}_j}$. Expressing $k(L)$ in terms of the variables $\bar{o}_j$ and $\bar{r}_j$ leads to

$$k(L) = \sum_{(i,j)\in R} \left(t_{i,j}\bar{r}_j - \bar{o}_j - \widehat{T}_i \right). \tag{15}$$

Similarly, the constraints (9) can be simplified to

$$\forall (i,j) \in R : t_{i,j}\bar{r}_j - \bar{o}_j - \widehat{T}_i \geq 0. \tag{16}$$

This is a linear objective function with linear constraints, which can be solved using standard linear program (LP) solvers like the simplex method.

For exponentially distributed errors, the maximum likelihood estimator is known to be nearly optimal. In our case, however, a different interpretation of the resulting approach is also possible. When comparing (12) and (7), we observe that

$$k(L) = \sum_{(i,j)\in R} \widehat{d}_{i,j}. \tag{17}$$

The optimal solution minimizes the sum of the estimated delays. Therefore, the resulting approach may also be understood as a form of constrained Least Absolute Deviation (LAD) regression. Since this interpretation is completely independent from the assumption of exponentially distributed delays, it supports the expectation that the derived estimator is also well-suited for delays with other distributions.

Note that the optimization problem (15) and (16) has the trivial solution $\forall j \in J : \bar{o}_j = \bar{r}_j = 0$ and $\forall i \in I : \widehat{T}_i = 0$. This is because, from the information contained in the log files, it is not possible to estimate all the absolute rates, but only the relative deviation between clocks. We call this the *rate ambiguity*. To overcome the rate ambiguity, we add a normalizing constraint $\sum_{j\in J} \bar{r}_j = |J|$; in the average, the inverse clock rates are assumed to be accurate. This assumption, however, is not crucial at all: if the average takes some other value, the solutions are simply scaled accordingly.

Similar to the rate ambiguity, there is also an *offset ambiguity* in the log files. The right hand sides of (15) and (16) do not change when all $\bar{o}_j$ are replaced with $\bar{o}_j + \tau$ and all $\widehat{T}_i$ are replaced with $\widehat{T}_i - \tau$, where $\tau \in \mathbb{R}$ is a given constant term. Thus, like above for the rates, it is not possible to estimate absolute, but only relative event times and clock offsets (even ignoring the fact that there is, of course, no "absolute time"). We may set, without loss of generality, $\bar{o}_1 = 0$.

If a reference clock is available—e. g., because at least one node has a connection to an external time source like a GPS receiver and records appropriate data—absolute synchronization to this reference is possible. More specifically, if the correct, global time of one event occurrence in one single node is known, then the offset ambiguity can be overcome. If the global times of two events, or, alternatively, the time of one event and the rate of one node are known, then the rate ambiguity can likewise be eliminated. This is possible either by adapting the constraints for rates and offset, or by a respective transformation of the synchronization result.

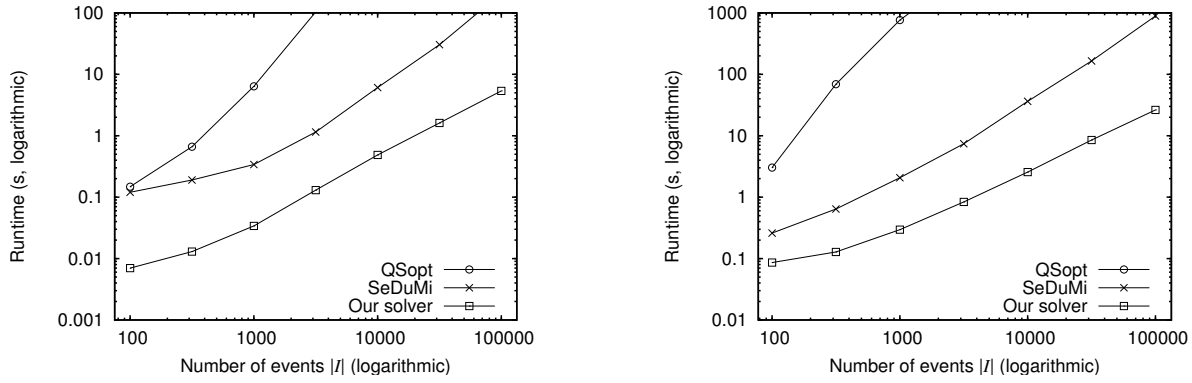
The resulting linear program can be written in the form

$$\text{minimize} \quad b^{\mathsf{T}} y \quad \text{subject to} \quad A^{\mathsf{T}} y \leq c, \tag{18}$$

where y is the vector of the unknowns $\widehat{T}_i$ for $i \in I$, followed by the vectors $\bar{o} \in \mathbb{R}^{|J|}$ and $\bar{r} \in \mathbb{R}^{|J|}$ of the $\bar{o}_j$ and $\bar{r}_j$ for $j \in J$, i. e.,

$$y = \begin{pmatrix} \widehat{T} \\ \bar{o} \\ \bar{r} \end{pmatrix}. \tag{19}$$

The matrix A^{T} represents the inequality constraints (16) and the normalizing constraints.



(a) $|J| = 20$. (b) $|J| = 100$.

Fig. 1. Performance comparison of QSopt, SeDuMi, and our own implementation as time synchronization LP solvers.

Events that have only been observed by one single node do not contribute information for the synchronization. Therefore, to keep the size of the linear program as small as possible, they should not be included in the optimization. Corrected timestamps for such events can easily be generated based on the rate and offset estimates.

V. Solving the Optimization Problem

In (18), (19) the maximum likelihood estimator is defined as the solution of a linear program with $|I| + 2 \cdot |J|$ variables and $|R|$ linear inequality constraints. Due to the size of the linear program a straightforward application of the simplex method may result in a significant effort in terms of computational power and memory. When solving (18) with a standard simplex solver like QSopt [16] the program takes hours to terminate even for relatively small problems. Therefore, we will now focus on the special structure of the linear program (18) and how it can be exploited to allow for a fast numerical solution. Below we outline the ideas behind our implementation of the synchronization approach. It is able to solve the linear program for data sets with $|J| \approx 100$, $|I| \approx 10^5$, and $|R| \approx 10^6$ on a standard PC within a few seconds.

Each row of A^T corresponding to a constraint (16) has exactly three non-zero entries and A is thus very sparse. The matrix A^T is closely related to the matrices arising in network optimization problems. In particular, it does not have full column rank. In the previous section, the offset ambiguity was introduced. Since we set $\bar{o}_1$ to zero, the corresponding column of A can be eliminated prior to the optimization. Our implementation checks for further redundancies that depend on the particular instance R and eliminates additional linearly dependent columns of A^T if existent.

We use a modern interior-point algorithm for our solver, a variant of Mehrotra's predictor corrector algorithm [17] that is particularly well suited to handle the structure of (18). The primary advantage of interior-point algorithms versus the simplex method is that interior-point methods do not suffer from degeneracy of the problem. Practical implementations very rarely take more than 70 to 100 iterations to solve a linear program. In our case, the particular structure of (18) can be exploited, making a single iteration very cheap. The concept of the algorithm as implemented here is based on Algorithm 14.3 in [18].

Apart from minor adjustments of the parameters proposed in [18], the main modification in our implementation concerns the storage format for the matrix A. Storing A directly would be extremely inefficient in terms of memory requirements as well as from a computational perspective. Our implementation comprises a specialized storage format for A, tailored to both the problem structure and the specific operations that appear in the interior-point algorithm. For matrices A arising from (18) this is a superior alternative to general purpose sparse matrix formats, as they are readily provided, e. g., by Matlab.

The main computational effort at each iteration of an interior-point algorithm is the computation and the Cholesky factorization of the matrix product $H = ADA^\mathsf{T}$. Here, D is a positive definite diagonal matrix that changes at each iteration. Due to our choice of setting up the variable y by first including $\widehat{T}$ and then $\bar{o}, \bar{r}$, the leading $|I| \times |I|$-block of H is a positive definite diagonal matrix; only the trailing $2 \cdot |J|$ rows and columns of H do have fill-in. This sparsity structure is also inherited by the Cholesky factor L of H. The leading $|I| \times |I|$-block of L can thus be computed in linear time. Given that typically $|I| \gg 2 \cdot |J|$, the computation of L and thus the solution of the overall problem is very cheap.

To demonstrate the huge gain in performance that is possible by using the tailored solver, we compare the runtime of our implementation with that of QSopt [16] and SeDuMi [19]. QSopt is, as mentioned before, a solver that uses the simplex method. SeDuMi on the other hand is a Matlab interior-point code that, like our own solver, benefits from the special structure of H, but uses a more general—and therefore somewhat slower—storage format of the sparse matrix A^T, and a more general sparse Cholesky factorization.

Figure 1 shows the computation times for calculating the solutions of optimizations with 20 nodes and with 100 nodes, for different numbers of shared events. All measurements have been made on an AMD Athlon X2 BE-2300 CPU with 1900 MHz and 1 GB of main memory. From the figure it can be seen that our implementation actually works very well. The tailored solver brings a large performance gain—it reduces the

computation time by typically at least a factor of 10–15.

Note that SeDuMi expects readily preprocessed input data in Matlab's sparse matrix format. The time needed for converting the data to this format is not included in the SeDuMi results in Figure 1. Especially for the larger problems, this conversion can take substantially longer than solving the problem. The processing times shown for our own solver do include the time for reading the input data and preparing the optimization problem. For our specialized matrix format this step can be performed very efficiently; it accounts only for a negligible fraction of the total processing time.

In particular the results with QSopt underline that an off-the-shelf simplex solver is in fact highly unsuitable for the specific type of linear optimization problem that we deal with. Not only does the computation time grow rapidly with an increasing problem size, but also do the memory requirements. By contrast, our tailored implementation is very memory efficient, and its observed runtime increases approximately linearly with the number of events $|I|$.

VI. Properties of the MLE

Now that we have seen that it is in fact possible to calculate a solution of the linear program and thus the maximum likelihood estimator within reasonable time, we are interested in the quality of this solution. In this section, we will thus tackle the question of how good the synchronization result is.

The available amount of data to estimate the clock deviations increases with an increasing number of network packets that have been received by multiple network nodes. Thus, intuitively, one could expect that the quality of the estimate improves with the availability of more experimental data. Similarly, it sounds reasonable that it is very unlikely that the result of the time synchronization process is grossly wrong if the input data is very accurate. In this section, we confine ourselves to a simplified variant of our estimator. For this simplified variant we can prove that these intuitive expectations are actually true. Since the complete proofs of these properties are quite complex and technical despite this simplification, we have not included them here. Instead they can be found in [20]. Below we will discuss the results and their implications, and we give a rough sketch of the proofs' ideas. Our numerical results presented later underline that the results also hold for the fully featured estimator with clock rate estimates.

The simplified estimator does not take clock rate deviations into account, i. e., it assumes that for each node the clock rate r_j is (approximately) 1 and thus correct with respect to the "true clock". Under this assumption the recorded time for a node-event pair $(i,j) \in R$ becomes $T_i + d_{i,j} + o_j$. Thus, the simplified maximum likelihood estimator, in analogy to the fully featured version, is the solution to the following problem:

$$\begin{aligned} &\text{maximize } L = \prod_{(i,j)\in R} \lambda e^{-\lambda\left(t_{i,j}-\widehat{o}_j-\widehat{T}_i\right)} \\ &\text{subject to } \forall (i,j) \in R : \widehat{d}_{i,j} = t_{i,j} - \widehat{o}_j - \widehat{T}_i \geq 0. \end{aligned} \tag{20}$$

The optimum is again independent of λ and the above is equivalent to

$$\text{minimize } k(L) = \sum_{(i,j)\in R} \left(t_{i,j} - \widehat{o}_j - \widehat{T}_i\right) = \sum_{(i,j)\in R} \widehat{d}_{i,j} \tag{21}$$

under the same constraints.

Here we will point out two desirable properties for this version of the estimator. First, we give tight error bounds on the estimation error that hold under the assumption of a bounded timestamping delay. In particular this means that the algorithm does not amplify errors. Furthermore, we show that the estimator is consistent; in other words, for increasing data set sizes the estimate converges (in probability) to the true values of the estimated features. It thus supports the intuition that the estimate improves for a larger amount of observed and logged events in the nodes.

A. Error bounds

In order to be able to give a bound for the estimation error, we need to make two additional assumptions. While the first guarantees network connectivity, the second establishes an upper bound on the timestamping delay. Note that an upper bound for the timestamping delay does not constrain the practical applicability of the results presented here: for any practical experiment there is a finite number of receptions, and thus also a maximum timestamping delay.

The existence of the offset ambiguity as introduced in Section IV prohibits that the absolute event times and clock offsets are determined from the log files. This also holds for the simplified estimator considered here. From the offset ambiguity, it is easy to see that there is also no way to estimate the relative time between two separate partitions within the same experiment. If there are no anchor points between two sets of nodes, there will be an ambiguity of the offset between these partitions. Thus, in order to get a bounded maximum estimation error, we need to assume network connectivity in the sense of anchor points: the network nodes do not fall into disjoint partitions, between which no events are shared.

Under such an assumption we can prove that

$$\forall j_1, j_2 \in J : |(o_{j_1} - o_{j_2}) - (\widehat{o}_{j_1} - \widehat{o}_{j_2})| \leq (|J|-1) \cdot D \tag{22}$$

if $D \in \mathbb{R}^+$ is an upper bound for the delays, i. e.,

$$\forall (i,j) \in R : d_{i,j} \leq D. \tag{23}$$

Note that the bound is on the difference between two estimation errors because of the offset ambiguity.

The basic idea of the proof is the following. Consider two nodes j_1 and j_2. It can be shown that a sequence of distinct nodes $s_1, ..., s_n$, $2 \leq n \leq |J|$, $s_1 = j_1$, $s_n = j_2$ with a special property always exists. In this sequence, for each pair of subsequent nodes s_q and s_{q+1}, there is an event observed by both s_q and s_{q+1}, for which the estimated timestamping delay in s_q is zero. This, together with the nonnegativity of the timestamping delays, allows for the construction of an upper bound for $(o_{j_1} - o_{j_2}) - (\widehat{o}_{j_1} - \widehat{o}_{j_2})$. Since j_1 and j_2 can be chosen arbitrarily, the same bound also holds with j_1 and j_2

interchanged. This yields a corresponding lower bound for j_1 and j_2 and thus constrains the absolute value as given in (22).

We are also able to show that under the mentioned assumptions the bound is the best possible, i. e., that no estimator can exist that achieves a smaller worst-case error. This proof is based on two explicitly constructed worst-case scenarios that result in identical log files. The point is that although the resulting local log files are identical for the two scenarios, the clock offsets differ so much that no estimate can be better than the worst-case bound given above in both cases.

From the bound on the clock offset estimation error it is then quite easy to come to a similar bound on the event time estimates:

$$\forall i_1, i_2 \in I : |(T_{i_1} - T_{i_2}) - (\widehat{T}_{i_1} - \widehat{T}_{i_2})| \leq |J| \cdot D. \quad (24)$$

No part of the proof exploits the exponential distribution of the delays. Thus, independent of the derivation of the estimator, it shows that if there is an upper bound for the timestamping delays, the estimates are close to the real values, regardless of the distribution of the delays within $[0, D]$.

B. Consistency

Differing from the results presented so far we will now no longer assume an upper bound on the timestamping delays. Instead we exploit their assumed exponential distribution. Under these premises, consistency of the simplified estimator can be established, which means convergence in probability to the correct offset values for an increasing number of observed events:

$$\forall j \in J : \operatorname*{plim}_{|I| \to \infty} \widehat{o}_j = o_j + x, \quad (25)$$

where $x \in \mathbb{R}$ again comes from the offset ambiguity.

Similar to the previous results it is clear that such a result cannot hold if the network is not connected. We show the consistency of the simplified MLE under an additional regularity condition, defined as follows. We say that this regularity condition is fulfilled if there exists an undirected, connected graph $G = (J, V)$ and some positive constant β such that

$$\forall \{j_1, j_2\} \in V : E\left[\left|\{i \in I | \{j_1, j_2\} \subseteq R_i\}\right|\right] \geq \beta \cdot |I|. \quad (26)$$

This precondition can be seen as a somewhat stronger variant of the connectivity assumption used above. It is stronger in the sense that it requires an (in expectancy) ever-growing number of independent connections between all parts of the network with an increasing total number of observed events.

In order to prove the consistency result we show that the probability of the likelihood function having its optimum in an arbitrarily small environment around the correct clock offset estimates is arbitrarily high for a sufficing number of observed events. The key idea is to introduce a per-event decomposition of the objective function $k(L)$. Certain properties of these event-wise objective function terms form the basis of our proof. We have seen before that $k(L)$ is simply the sum of the $\widehat{d}_{i,j}$ for all (i, j) in R. Then a decomposition of $k(L)$ into event-wise components f_i is trivial:

$$f_i := \sum_{j \in R_i} \widehat{d}_{i,j} \qquad k(L) = \sum_{i \in I} f_i. \quad (27)$$

We then switch our point of view. We regard the f_i no longer as functions of the estimated latencies, but as functions of the estimation error. It is then quite straightforward to show that all the f_i are convex and that they are all Lipschitz continuous with a common Lipschitz constant. Furthermore, we show that the expectancy for each f_i—as a function of the estimation error—has a global minimum for the correct estimate, and we give a non-negative lower bound for the difference between this expectancy in case of a non-zero estimation error and the minimum value. All these results in conjunction with the law of large numbers can then be used to establish the consistency of the estimator: for a given $\delta > 0$ there is a number of events N such that for $|I| > N$ the probability that the estimation error is greater than δ becomes arbitrarily small.

From the consistency result for the clock offset estimate, it is easy to obtain a result on the quality of the event time estimates in the same asymptotic setting. If the estimation error of the clock offsets is close to zero (neglecting the offset ambiguity), the remaining event time error for an event i is $\min_{j \in R_i} d_{i,j}$. This minimum of the independent, exponentially distributed $d_{i,j}$ is itself exponentially distributed with parameter $|R_i| \cdot \lambda$. In particular this means that the expected estimation error decreases with the number of nodes observing the same event.

VII. Numerical Evaluation

While the previous section assessed the performance of the proposed time synchronization method analytically, we will now focus on numerical experiments with the algorithm. In particular, we will show that the asymptotic properties that have been proven for the simplified estimator hold also for the fully featured version with clock rate estimates. Moreover, it will become clear that the convergence is quick enough to yield accurate estimates even for small event counts. Finally, we will show that the algorithm is robust if the assumptions—in particular the exponential distribution of the timestamping delays and the negligibility of clock drift—do not hold.

A. Methodology

Although desirable, using log files from a real testbed for an evaluation that rigorously quantifies the numerical quality of the calculations and the convergence speed is not possible: for real hardware, the correct values for the rates, offsets, and event times cannot be determined—this is why we need post-experiment time synchronization in the first place. Therefore, we use a two-step simulation in which the correct values are known. In the first step, the network is simulated to obtain globally consistent event times and a receiver relation R. Then, subsequently, we simulate the timestamping of the events in each node. Random clock rates, offsets, and timestamping delays are used to transform the correct timestamps, yielding a set of per-node log files. Like after a real experiment, our algorithm is then given these log files as input. The quality of the solution can be determined by comparing the results to the correct times, rates, and offsets.

How a simulation should be set up for credible network protocol evaluation results is a highly controversial and heavily discussed topic. Since our focus here, however, is on supplying

the numerical algorithm evaluation with an event set I, event times T, and a receiver relation R, and not on a realistic performance evaluation of some protocol, we constrain ourselves to a basic simulation scenario. We use the network simulator ns-2 [21], which has been extended to support promiscuous mode-like packet tracing: if a data packet could be successfully decoded by a node's simulated wireless interface, the packet is timestamped and logged, regardless of whether the node was the intended destination or just able to overhear the transmission.

In our simulations, $|J| = 100$ nodes move on an area of 1200×1200 meters according to the random waypoint mobility model. AODV [22] is used as a multihop routing protocol. Five pairs of nodes communicate continuously over a simulation time of 10 minutes, performing FTP data transfers over TCP connections. The IEEE 802.11 MAC protocol is used at a fixed network bandwidth of 1 MBit/s. The radio range is set to 250 meters, the carrier sense radius to 550 meters.

For the generation of the local node log files, the clock offset and rate of every node were chosen randomly. The choice of the offset is not at all critical: our implementation actually exploits the offset ambiguity to achieve improved numerical stability and, as its first step, shifts all processed log files to start at time zero. Consequently, whichever offset is chosen for a node, the performed calculations and thus the accuracy of the estimates are virtually identical. In our simulations, we sample the offsets from a normal distribution with mean zero and standard deviation five seconds. For the clock rates, we used a gamma distribution[2] with mean one and different standard deviations. The gamma distribution has the advantages of yielding only positive rate values, and being concentrated around the expectancy. The probability density function of a gamma distribution is depicted in Figure 2. Unless otherwise stated, the parameters have been chosen to yield a standard deviation of 100 parts per million (ppm), which is rather pessimistic, meaning that on average a clock is wrong by more than eight seconds per day.

To be able to compare the synchronization results directly with the correct values from the simulation trace file, the rate and offset ambiguities need to be overcome. As stated in Section IV, the normalization constraints lead to scaled and shifted results if the average inverse clock rate is different from 1, and if the offset of node 1 is different from 0. This scaling and shifting can easily be removed by a linear-affine transformation after the optimization, based on the average inverse clock rate in the simulation and the offset chosen for the first node.

Because ns-2 simulates the radio propagation delay, the arrival times of the same packet at different nodes actually differ slightly. For calculating the event time errors, we compare the estimated event time to the average ns-2 reception time. The differences are in the order of 10^{-7} seconds, and therefore significantly below the other errors that we are dealing with here.

[2]The gamma distribution is given by the probability density function

$$f(x; k, \theta) = x^{k-1} \frac{e^{-x/\theta}}{\theta^k \cdot \Gamma(k)} \quad \text{for } x > 0,$$

where Γ is the gamma function. The gamma distribution has two parameters, called the *shape parameter* k and the *scale parameter* θ. It has mean $k \cdot \theta$ and variance $k \cdot \theta^2$.

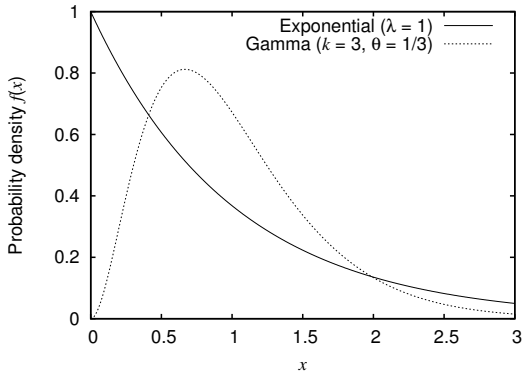


Fig. 2. Probability density functions of exponential distribution ($\lambda = 1$) and gamma distribution ($k = 3$, $\theta = 1/3$).

B. Convergence and Numerical Accuracy

In our first set of experiments, we simulated the timestamping delays according to our assumptions, i. e., exponentially distributed. We varied the expected timestamping delay λ^{-1} between 10^{-3} and 10^{-5} seconds, and increased the number of events used for the synchronization. One central result in the previous section was that—at least for the simplified estimator—we can expect the quality of the estimates to improve if an increasing number of events is available for synchronization. In a practical implementation, numerical effects of, e. g., a limited floating point precision may influence the results. The primary purpose of our first simulations is to verify that what we have shown in theory for the simplified case also holds for our practical implementation of the full estimator, and also to give an idea of the convergence speed.

Figure 3 shows the resulting average event time error with 95-percentile error bars. For better readability of the chart, only the upper part of the error bars is shown. The x-axis denotes the number of events that have been used for the synchronization. These have been chosen randomly from all transmissions with more than one receiver. Note that at the left hand side of the chart, for 100 events, there is only one sent packet per node on average. The randomly chosen clock errors are quite significant. Still, the synchronization eliminates them to an extent that allows for accurate event time estimates. If more events are available for the synchronization, the estimates improve further quickly, and the average event time errors are one order of magnitude below the timestamping delays. The convergence is so quick that for 1000 available anchor point events and more, only tiny fluctuations are left.

From these as well as from our other results, it can also be seen that the extent and the nature of the timestamping delays constitute the central limiting factor for the achievable event time estimate accuracy. This is a common limitation to any synchronization solution. The error of the rate and offset estimates, however, tends to zero for large $|I|$. This is evident from Figure 4, which shows how the rate error develops in the

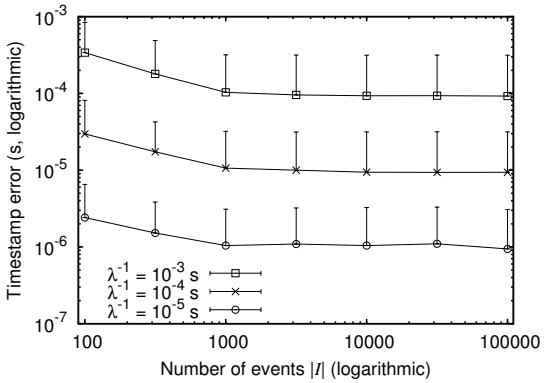


Fig. 3. Event time errors depending on the number of events $|I|$ and the average timestamping delay λ^{-1}.

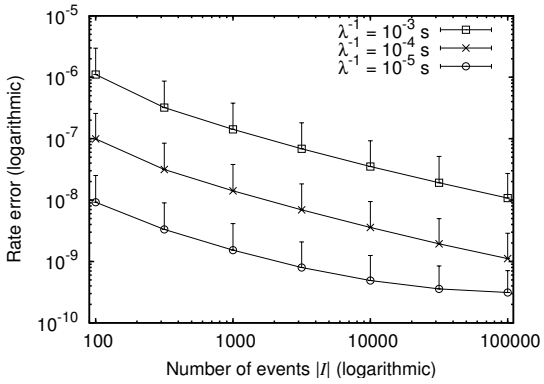


Fig. 4. Rate errors depending on the number of events $|I|$ and the average timestamping delay λ^{-1}.

same setting as before. The figure also displays the different levels of accuracy depending on the order of magnitude of the timestamping delays. The corresponding results for the offset estimates show the same behavior. For very small delays with $\lambda^{-1} = 10^{-5}$ seconds and a high number of available anchor points, it can be seen that the accuracy does no longer improve linearly; then the implementation approaches its numerical accuracy limits.

It also turns out that the estimation errors of rate and offset are largely independent from the true values of these parameters and their spread. For the offsets, this is clear from the problem structure. For the rates, however, this trait is not immediately obvious. Tables I and II show the accuracy of the estimation results. The small deviations in estimation accuracy are remaining statistical fluctuations.

Another theoretical result from the previous section is that the event time estimation accuracy for an event i increases

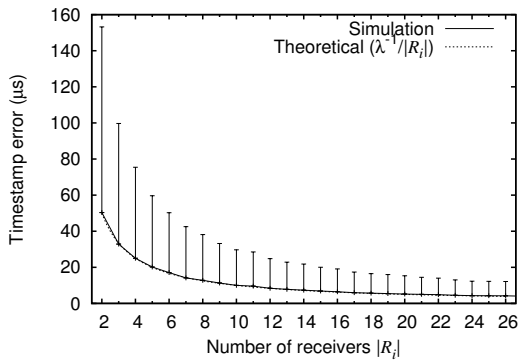


Fig. 5. Theoretical and simulated event time estimation errors depending on the number of receivers $|R_i|$ ($\lambda^{-1} = 10^{-4}$ s, $|I| = 10\,000$).

with $|R_i|$. More specifically, the result said that for correct rate and offset estimates, the remaining expected event time error is exponentially distributed with parameter $|R_i| \cdot \lambda$, and thus is $\lambda^{-1}/|R_i|$ on average. Figure 5 shows simulation results from log files with $\lambda^{-1} = 10^{-4}$ seconds and $|I| = 10\,000$. They exhibit exactly the predicted behavior. The chart shows the average event time error and again the 95-percentile upper error bar, where the events are broken down along the x-axis according to the number of nodes $|R_i|$ that observed them. The chart shows also the theoretical average error given by the function $x \mapsto \lambda^{-1}/x$, and it is evident that the results match the theoretical expectations very closely.

We conclude that the convergence of the estimate is very quick, and a reasonable synchronization quality can be expected even if only a limited number of anchor points is available. The results also underline that the numerical performance of our implementation will not be the limiting factor for the accuracy in practical usage.

C. Robustness

So far, our simulations have used clocks and timestamping delays that match the assumptions made for the derivation of the approach. Now we assess how robust the estimator is if these assumptions do not hold. We thus use the very same estimator as before, but generate simulation data that intentionally contradicts the assumptions in different ways.

The event time estimation errors occurring in these experiments are shown in Figure 6. The results with $\lambda^{-1} = 10^{-4}$ seconds from the previous simulations, where all our assumptions hold, are used as a "baseline" for comparison (labeled "exponential"). Since the error bars in particular are difficult to identify in this figure, Table III provides a more detailed view on the exact values for $|I| = 10\,000$. As could

TABLE I

CLOCK RATE ESTIMATION ERROR FOR DIFFERENT CLOCK RATE STANDARD DEVIATIONS ($\lambda^{-1} = 10^{-4}$ s, $|I| = 10\,000$).

std. dev. of rates	avg. rate error	95-perc. max rate error
10 ppm	0.00352 ppm	0.00935 ppm
100 ppm	0.00358 ppm	0.00945 ppm
1000 ppm	0.00355 ppm	0.00927 ppm

TABLE II

CLOCK OFFSET ESTIMATION ERROR FOR DIFFERENT CLOCK RATE STANDARD DEVIATIONS ($\lambda^{-1} = 10^{-4}$ s, $|I| = 10\,000$).

std. dev. of rates	avg. offset error	95-perc. max offset error
10 ppm	1.56 μs	3.84 μs
100 ppm	1.50 μs	3.81 μs
1000 ppm	1.50 μs	3.96 μs

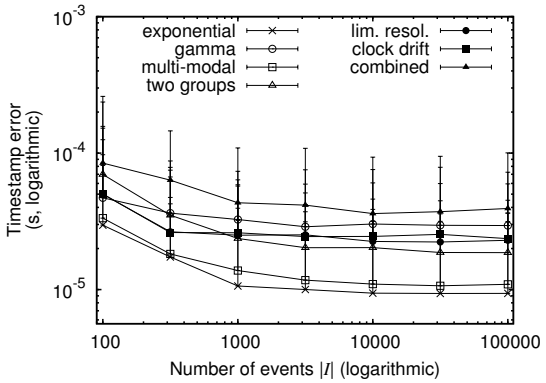


Fig. 6. Event time estimation errors for increasing $|I|$ if assumptions do not hold.

TABLE III
EVENT TIME ESTIMATION ERRORS FOR $|I| = 10\,000$ EVENTS IF ASSUMPTIONS DO NOT HOLD.

distribution	avg. timestamp err.	95-perc. max timestamp err.
exponential	9.4 μs	31.6 μs
gamma	30.2 μs	60.5 μs
multi-modal	11.0 μs	35.9 μs
two groups	20.3 μs	38.7 μs
lim. resol.	22.5 μs	46.0 μs
clock drift	24.5 μs	75.8 μs
combined	36.0 μs	93.5 μs

be expected, the estimation results with exponential delays are slightly better than those where the assumptions do not hold.

First, we varied the distribution from which the timestamping delays were sampled. The result labeled "gamma" shows the estimation error for delays drawn from a gamma distribution with shape parameter $k = 3$. The scale parameter θ has been set to $1/(k \cdot \lambda)$. This yields a mean of λ^{-1} and therefore allows for a direct comparison to the results with exponentially distributed delays with the same mean. The probability density functions of these two distributions are shown in Figure 2, both adjusted to mean 1.

In the "multi-modal" simulations we assess the robustness to outliers in the timestamping delays. The majority of timestamping delays follows an exponential distribution with $\lambda^{-1} = 10^{-4}$ seconds. 10 %, however, are instead drawn from a gamma distribution with $k = 10$ and a mean ten times higher. 1 % are "heavy outliers", sampled from a gamma distribution with a mean 50 times higher and $k = 100$. From these results, it can be seen in particular that our proposed method is very robust against outliers.

Heterogeneous hardware with different timestamping delay characteristics is simulated in the "two groups" setup. The simulated network nodes are divided into two groups of 50 nodes each. The delays are exponentially distributed here, but the values of λ^{-1} differ by one order of magnitude: one half of the nodes uses $\lambda^{-1} = 10^{-4.5}$ seconds, and the other half uses $\lambda^{-1} = 10^{-3.5}$ seconds.

We have also simulated the effects of a bad clock accuracy. As stated earlier, clocks in computer systems sometimes have a rather coarse resolution. In the simulations labeled "lim. resol.", the timestamping delays are again exponentially distributed with $\lambda^{-1} = 10^{-4}$ seconds, but the timestamps' resolution has been reduced to 0.1 milliseconds prior to performing the time synchronization. The estimator again deals very well with this effect. It is particularly remarkable that the availability of measurements from multiple nodes with different offsets allows for an estimation of the event times that is more accurate than the resolution of a single node's clock.

The "clock drift" simulations show the effect that randomly drifting clocks have on the accuracy of the estimates. Instead of linear clock functions, we use second order polynomials. Clock drifts are chosen independently from a Gaussian distribution with mean zero and standard deviation $3 \cdot 10^{-9}$. With this, the speed change of a clock can easily sum up to several ppm during a ten-minute simulation. Nevertheless, as our results show, the effect on the synchronization accuracy is very limited.

Finally, the "combined" simulations incorporate all of the above sources of inaccuracies. In this data set, the timestamps are delayed according to the outlier-prone "multi-modal" distribution, there are two groups of nodes with different expected timestamping delays like in the "two groups" simulations, the simulated clocks drift as described above, and the timestamps' resolution is again limited to 0.1 ms. Even this combination of effects—all of which heavily contradict the foundations on which we have initially built our method—results in a degradation of the estimation quality by substantially less than one order of magnitude.

In summary, the estimator has proven to be very robust and yields sensible results also if the various assumptions made for its derivation do not exactly hold. The nature of the timestamping delays clearly impacts the achieved precision. But although the quality of the estimates may degrade to a certain extent (as could be expected), the errors are still small, and the estimator converges quickly to a high accuracy in all cases.

VIII. Real-World Experiments

The previously presented robustness assessment has shown that our proposed time synchronization method is able to deal well with a whole range of adversarial effects in the log data. Still, however, these evaluations were based on artificially generated simulation data. We will thus now complement them with an application of our method to real-world experimental data. Due to the unknown true values this does not allow to rigorously determine the remaining errors. But it nevertheless provides a good intuitive understanding and shows how well the method can handle real data.

Our experimental setup consist of seven PCs with rather heterogeneous hardware both in terms of CPU/memory and the wireless interface card. One of these nodes periodically broadcasts one packet per second, over a total of 20 minutes. The other six record and timestamp the received packets. Initially, the offsets have been reduced by approximately setting the clocks by hand.

In our figures, we use one of the receivers as a reference, and plot the differences in the recorded timestamps between

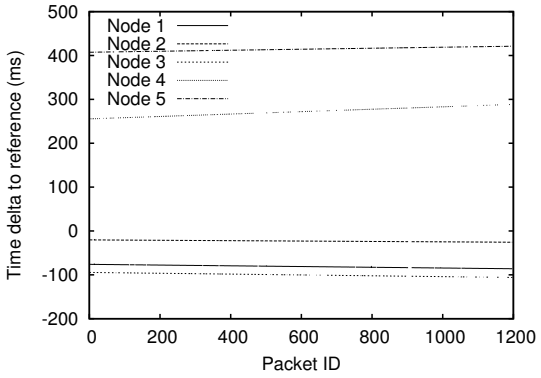


Fig. 7. Unsynchronized timestamp differences in real-world experiments.

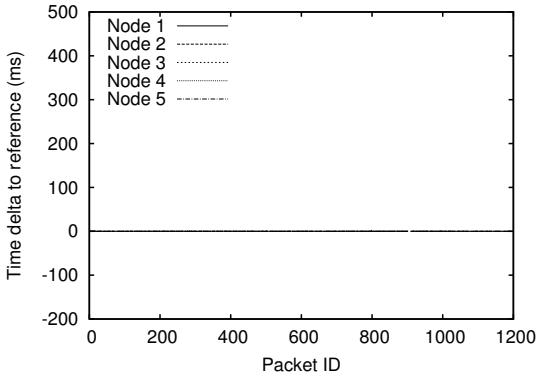


Fig. 8. Synchronized timestamp differences in real-world experiments.

this receiver and the other five. Figure 7 shows how—for unsynchronized clocks—this difference develops during the experiment. The almost exactly linear relative clock errors are clearly visible. Where the lines in the plot are interrupted, the respective nodes have missed packets.

We have used our time synchronization algorithm on the data from the above experiment. This yields estimates $\widehat{r}_j$ and $\widehat{o}_j$ for the rates and offsets of the six receivers. We use those to eliminate the estimated linear clock deviations from Figure 7, by computing

$$\frac{t_{i,j} - \widehat{o}_j}{\widehat{r}_j} \approx T_i + d_{i,j} \tag{28}$$

for each timestamp. In Figure 8, we show the results of this correction. Again we plot the timestamp differences to the reference node, the y-axis uses the same scale as in Figure 7.

The approximation in (28) is exact if the estimates $\widehat{r}_j$ and $\widehat{o}_j$ of r_j and o_j are exact. Remaining clock deviations or estimation errors would therefore be visible in Figure 8: they would result in remaining timestamp differences to the reference node.

That such errors are in fact virtually non-existent becomes clear if we zoom the y-axis further in, as we do in Figure 9. It can be seen that the timestamping differences are typically in the order of some ten microseconds, with occasional outliers of up to 1–2 milliseconds. There is, however, no sign of a systematic (i. e., rate or offset estimation) error, like clocks drifting apart over time. This indicates that the rate and offset

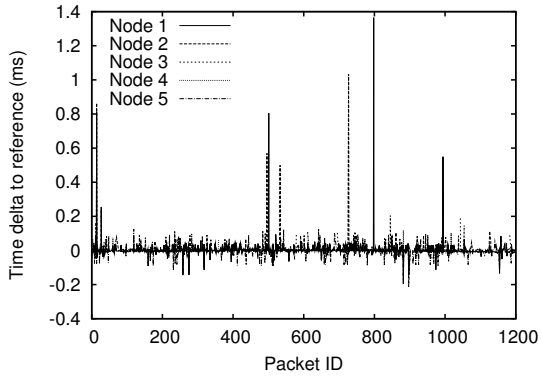


Fig. 9. Synchronized timestamp differences in real-world experiments (zoomed).

estimates are indeed correct.

Note that eliminating the estimated linear clock deviations according to (28) leaves the timestamping delays in the data. In a practical application, our approach would have been able to also eliminate long timestamping delays with very high probability. Recall that given exact rate and error estimates, it removes all but the shortest timestamping delay that occurred for events with multiple observers. Since we do not know the true times of the packet receptions in the experiment, we cannot tell how large exactly the then remaining deviations are. It seems reasonable, however, to assume that the smallest timestamping delay for a packet is within the same order of magnitude as the minimal timestamp difference to the reference node[3]. Considering (28) this difference is simply the difference of two timestamping delays. In the discussed experiment, the average of the per-packet minimum timestamp difference is 5 microseconds; for 95 % of the packets, it is below 29 microseconds.

In order to examine the impact of clock drift, we have performed a second experiment with a duration of 100 minutes. Note that this significantly exceeds the time span over which drift can be neglected. The experimental setup as

[3]This does of course not hold when the minimum path delay is included in the timestamping delay. This, however, is not a problem here: recall from Section III-C that the minimum path delay in a node is equivalent to an additional clock offset, and may thus be eliminated.

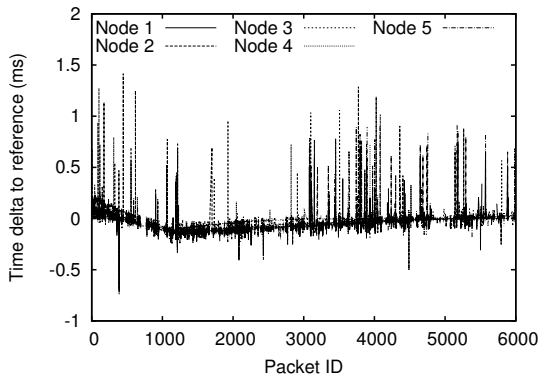


Fig. 10. Synchronized timestamp differences in a real-world experiment with an overall duration of 100 minutes.

well as the used evaluation methodology are the same as above. The results of this experiment after correcting the timestamps according to (28) are shown in Figure 10. The non-linear deviations—effects of clock drift—are clearly visible. Nevertheless, our approach provides good estimates for the offsets and rates. Still, 95 % of all timestamp differences are below 142 microseconds.

IX. Conclusion

In this paper we have considered offline time synchronization for networks with local broadcast media. We have proposed a method to combine separate event log files from nodes in such a network into one single log file with a common time basis, in spite of deviating local clocks and latencies that occur when the timestamps for the events are generated. This is useful, for example, for the evaluation of experimental results. The key issue is how the deviations of the clocks and the latencies can be addressed without the necessity of additional communication between the network nodes.

Our algorithm utilizes transmissions that have been received by multiple nodes as anchor points. It has been shown how the synchronization can be formulated as an optimization problem, and how this problem can be expressed as a linear program. The special structure of this linear program can then be exploited by an efficient solution algorithm. We have presented an implementation of a specialized solver, and comparisons to other LP solvers underline the performance of the employed solution techniques. Furthermore, we have presented analytical results on the quality of the synchronization for a simplified variant of the estimator. In particular, these results include a bound on the maximum possible synchronization error, depending on the maximum timestamping delay, and a consistency proof. A subsequent numerical evaluation shows that the convergence to accurate estimates is quick, and that the solver is robust even if the underlying assumptions do not hold. Finally, it has been shown in a number of experiments that the estimator is able to correctly handle real-world data in the presence of timestamping delays and clock drift.

We consider the presented approach generally applicable whenever event data is distributed over multiple sources, and common events can be used as anchor points for a maximum likelihood time synchronization. Apart from supporting the interpretation of experimental results in networks with local broadcast media that we have primarily considered, we envision adaptations of the proposed technique for example in the field of network forensics, where data of, e. g., multiple intrusion detection systems (IDS) or firewall logs is combined. There, too, certain events will often have been observed in parallel by multiple systems.

Acknowledgments

The authors would like to thank the German Research Foundation (DFG) for partially funding this research.

References

[1] D. L. Mills, “Internet time synchronization: The network time protocol,” in *Global States and Time in Distributed Systems*, Z. Yang and T. A. Marsland, Eds. IEEE Computer Society Press, 1994.

[2] D. Mills, “Network Time Protocol (Version 3) Specification, Implementation and Analysis,” RFC 1305 (Draft Standard), Mar. 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1305.txt

[3] K. Römer, P. Blum, and L. Meier, “Time synchronization and calibration in wireless sensor networks,” in *Handbook of Sensor Networks: Algorithms and Architectures*, I. Stojmenovic, Ed. John Wiley & Sons, Sept. 2005, pp. 199–237.

[4] P. Veríssimo, L. Rodrigues, and A. Casimiro, “Cesiumspray: a precise and accurate global time service for large-scale systems,” *Real-Time Systems*, vol. 12, no. 3, pp. 243–294, 1997.

[5] M. Mock, R. Frings, E. Nett, and S. Trikaliotis, “Continuous clock synchronization in wireless real-time applications,” in *SRDS ’00: Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, Oct. 2000, pp. 125–132.

[6] J. Elson, L. Girod, and D. Estrin, “Fine-grained network time synchronization using reference broadcasts,” in *OSDI ’02: Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2002, pp. 147–163.

[7] R. M. Karp, J. Elson, C. H. Papadimitriou, and S. Shenker, “Global synchronization in sensornets,” in *LATIN ’04: Proceedings of the 6th Latin American Symposium on Theoretical Informatics*, Apr. 2004, pp. 609–624.

[8] A. Duda, G. Harrus, Y. Haddad, and G. Bernard, “Estimating global time in distributed systems,” in *ICDCS ’87: Proceedings of the 7th International Conference on Distributed Computing Systems*, Sept. 1987, pp. 299–306.

[9] P. Ashton, “Algorithms for off-line clock synchronization,” Department of Computer Science, University of Canterbury, Tech. Rep. TR COSC 12/95, December 1995.

[10] S. B. Moon, P. Skelly, and D. F. Towsley, “Estimation and removal of clock skew from network delay measurements,” in *INFOCOM ’99: Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, Mar. 1999, pp. 227–234.

[11] L. Zhang, Z. Liu, and C. H. Xia, “Clock synchronization algorithms for network measurements,” in *INFOCOM ’02: Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies*, June 2002, pp. 160–169.

[12] D. W. Allan, “Time and frequency (time-domain) characterization, estimation, and prediction of precision clocks and oscillators,” *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. UFFC-34, no. 6, pp. 647–654, Nov. 1987.

[13] W. Kiess and M. Mauve, “Real-world evaluation of mobile ad-hoc networks,” in *Multi-hop Ad hoc Networks from Theory to Reality*, M. Conti, J. Crowcroft, and A. Passarella, Eds. Hauppauge, NY, USA: Nova Science Publishers, 2007, pp. 1–22.

[14] D. Veitch, S. Babu, and A. Pàsztor, “Robust synchronization of software clocks across the Internet,” in *IMC ’04: Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, Oct. 2004, pp. 219–232.

[15] H. Kopetz and W. Ochsenreiter, “Clock synchronization in distributed real-time systems,” *IEEE Transactions on Communications*, vol. 36, no. 8, pp. 933–940, 1987.

[16] D. Applegate, W. Cook, S. Dash, and M. Mevenkamp, “QSopt linear programming solver,” version 1.01. [Online]. Available: http://www2.isye.gatech.edu/~wcook/qsopt/

[17] S. Mehrotra, “On the implementation of a primal-dual interior point method,” *SIAM Journal on Optimization*, vol. 2, no. 4, pp. 575–601, 1992.

[18] J. Nocedal and S. J. Wright, *Numerical Optimization*. Berlin: Springer, 1999.

[19] J. F. Sturm, O. Romanko, and I. Pólik, “SeDuMi,” version 1.1R2. [Online]. Available: http://sedumi.mcmaster.ca/

[20] B. Scheuermann, W. Kiess, M. Roos, F. Jarre, and M. Mauve, “Error bounds and consistency of maximum likelihood time synchronization,” Department of Computer Science, Heinrich Heine University Düsseldorf, Germany, Tech. Rep. TR-2008-001.

[21] “The ns-2 network simulator,” version 2.30. [Online]. Available: http://www.isi.edu/nsnam/ns

[22] C. E. Perkins and E. M. Royer, “Ad-hoc on-demand distance vector routing,” in *WMCSA ’99: Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, Feb. 1999, pp. 90–100.

Björn Scheuermann received the B. S. degree in Mathematics and Computer Science in 2004 and the M. S. in Computer Science in the same year, both from the University of Mannheim, Germany. He got a scholarship from the German National Merit Foundation. In 2007, he received his Ph. D. in Computer Science from the Heinrich Heine University, Düsseldorf, Germany, where he is currently a postdoctoral researcher. His research interests include mobile ad-hoc networks and vehicular ad-hoc networks.

Wolfgang Kiess studied computer science at the University of Mannheim, Germany, and the University of Nice Sophia Antipolis, France, from 1997 to 2003. He received the M. S. degree in 2003. Currently, he is a Ph. D. student at the Heinrich Heine University, Düsseldorf, Germany. His current research interests include mobile and vehicular ad-hoc networks with a focus on experimental evaluations.

Magnus Roos received the B. S. degree in Mathematics from the Heinrich Heine University, Düsseldorf, Germany, in 2006. The topic of his Bachelor's thesis was on the implementation of the time synchronization approach presented in this paper. Currently, he is a graduate student in Computer Science at the Heinrich Heine University.

Florian Jarre received the M. A. and Ph. D. degrees in Mathematics from the University of Texas at Austin in 1986 and from the University of Würzburg, Germany, in 1989. From 1989 to 1999, he was an Assistant Professor at the University of Würzburg. He was a visiting scholar at Stanford University (1990–1991), visiting professor at the Institute of Statistical Mathematics in Tokyo (1994–1995), and at the University of Trier, Germany (1998–1999). From 1999 to 2000 he was an Associate Professor at the University of Notre Dame (USA), and in 2000, he joined the Heinrich Heine University, Düsseldorf, Germany, as a Full Professor. His main research interest is mathematical optimization.

Martin Mauve received the M. S. and Ph. D. degrees in Computer Science from the University of Mannheim, Germany, in 1997 and 2000 respectively. From 2000 to 2003, he was an Assistant Professor at the University of Mannheim. In 2003, he joined the Heinrich Heine University, Düsseldorf, Germany, as a Full Professor and Head of the research group for computer networks and communication systems. His research interests include distributed multimedia systems, multimedia transport protocols, mobile ad-hoc networks and inter-vehicle communication.