Optimal Clock Synchronization in Networks

Christoph Lenzen
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
lenzen@tik.ee.ethz.ch

Philipp Sommer
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
sommer@tik.ee.ethz.ch

Roger Wattenhofer
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

Abstract

Having access to an accurate time is a vital building block in all networks; in wireless sensor networks even more so, because wireless media access or data fusion may depend on it. Starting out with a novel analysis, we show that orthodox clock synchronization algorithms make fundamental mistakes. The state-of-the-art clock synchronization algorithm FTSP exhibits an error that grows exponentially with the size of the network, for instance. Since the involved parameters are small, the error only becomes visible in mid-size networks of about 10-20 nodes. In contrast, we present *PulseSync*, a new clock synchronization algorithm that is asymptotically optimal. We evaluate PulseSync on a Mica2 testbed, and by simulation on larger networks. On a 20 node network, the prototype implementation of PulseSync outperforms FTSP by a factor of 5. Theory and simulation show that for larger networks, PulseSync offers an accuracy which is several orders of magnitude better than FTSP. To round off the presentation, we investigate several optimization issues, e.g. media access and local skew.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Architecture and Design, Network Protocols

General Terms

Algorithms, Design, Theory, Experimentation, Measurement, Performance

Keywords

Sensor Networks, Time Synchronization, Clock Drift, Lower Bound

1 Introduction

Without doubt, providing a common notion of time is one of the most basic services in any distributed system. Several applications depend on network nodes having a precisely synchronized time. In wireless sensor networks, accurate time is particularly important, first and foremost because of energy efficiency. In order to increase the network lifetime, nodes try to minimize their duty cycle by sleeping whenever possible. To get any work done, nodes must wake up from time to time, in order to communicate with their neighbors, for instance. Although one can imagine several ways to establish communication even without accurate clock synchronization, the most energy efficient protocols let two neighboring nodes wake up at (hopefully) exactly the same time. The neighbor nodes can make sure very quickly that no messages have to be exchanged, and go back to sleep. If they need to exchange information, they will go back to sleep as soon as finished.

However, having a precise common time has many applications beyond wireless media access and energy efficiency. Even in *wired* sensor networks one may be interested in having an exact time, since several nodes may experience the same acoustic or seismic event [2, 27, 32], and the better local times are synchronized, the better the location of such an event can be detected. Indeed, an improvement in time accuracy will linearly improve spatial resolution. Finally, accurate time will help in networks where nodes are neither wireless nor equipped with (acoustic) sensors, for instance in multiplayer Internet games.

How is an accurate common time achieved? One solution is to equip all nodes with receiver modules to have access to UTC provided by the Global Positioning System (GPS). In many applications this is impossible, e.g., in indoor applications, or just too expensive. Instead, each node is equipped with a cheap hardware clock which is not really accurate; over time it will accumulate drift. To reduce this drift, a *clock synchronization* algorithm is implemented: Nodes communicate by regularly exchanging messages, trying to minimize the synchronization error between the nodes in the network.

This is challenging due to several issues. Apart from the drifting clocks, synchronization is hindered by the fact that the time to send, receive, and process messages cannot be determined exactly. Furthermore, clock drifts change over time; the most prominent factor here is temperature (cf. Figure 1). This is bad news, since it imposes that a) nodes must periodically re-synchronize and b) the system has to detect variations in clock speeds and adapt quickly, both goals that contradict the desire to save energy. Therefore, an efficient protocol cannot simply enforce the best possible synchro-

SenSys'09, November 4–6, 2009, Berkeley, CA, USA.

nization; rather it has to optimize the trade-off between skew bounds and communication costs.

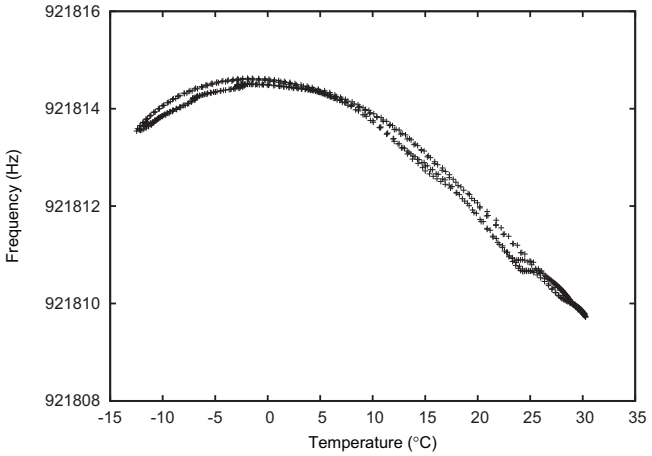


Figure 1: Hardware clock frequency of a Mica2 node for different ambient temperatures. A difference of 5 degrees alters the clock speed by up to one microsecond per second.

Over time a variety of clock synchronization algorithms have been proposed and implemented. A typical lecture on clock synchronization will cover the usual suspects, from NTP over RBS and TPSN to FTSP. Everybody teaching the same is a good sign of research being done and over. So, is clock synchronization *solved*?

We believe not. We start out with a theoretical analysis of clock synchronization, tailored to provide results having impact in practical scenarios. Analyzing the state-of-the-art protocol FTSP in this theoretical framework will reveal that it does have a serious scalability problem. It will only become visible in mid-size networks of about 10-20 nodes, but the problem exists, as clock skew in FTSP grows *exponentially* with network diameter. For many applications, e.g., the deployment at the Golden Gate Bridge [13] which has a network diameter of 46 hops, it is important to have precisely synchronized clocks even for far-away nodes in the network. Moreover, this example illustrates that wireless sensor nodes may be deployed in environments with rapidly changing ambient temperature. In such scenarios, a fast response to variations in clock drifts becomes a most pressing issue in the design of a suitable synchronization protocol.

Consequently, we are going to propose *PulseSync*, a novel clock synchronization algorithm that matches the lower bound emerging from the theoretical analysis. Time information from a reference node is propagated as fast as possible through the network, allowing for adaptivity to changes in network topology and temperature. Furthermore, we employ a sophisticated mechanism to prevent drift estimation errors from propagation through the network and impairing the synchronization accuracy. PulseSync improves the synchronization accuracy while using equal or less energy than FTSP and other current clock synchronization protocols.

We will back our theoretical results with measurements on a testbed of 20 Mica2 nodes, and with extensive simulations of larger networks. The experiments show an average network synchronization error of FTSP of 23.96 µs with a maximum error of 249 µs. PulseSync reduces this clock skew significantly to average and maximum skews of only 4.4 µs and 38 µs, respectively. The simulations forecast that this gap widens rapidly; for 50 nodes, FTSP will be several orders of magnitude worse, whereas clock skews of PulseSync will grow by less than a factor 2.

Finally, we discuss the issues of efficient media access and possible optimizations regarding the local skew.

2 Related Work

Clock synchronization has been studied extensively ever since the advent of distributed systems. In wireless networks, the classic solution is to employ an atomic clock, such as in the global positioning system (GPS). However, limitations in terms of cost and energy make it in general unfeasible to equip each sensor node with its own GPS receiver. Moreover, line of sight to the GPS satellites is needed, restricting the use of GPS to outdoor applications.

Classical clock synchronization algorithms rely on the ability to exchange messages at a high rate, which may not be possible in wireless networks, and certainly is not energy-efficient. Traditional algorithms like the *Network Time Protocol (NTP)* [21] are to complex to be used for sensor network applications. Moreover, as their original application domain is the Internet, they are not accurate enough for our purpose; even in a local area network they may experience a clock skew in the order of milliseconds.

To achieve considerable better results, sensor networks require sophisticated algorithms for clock synchronization since the hardware clocks in sensor nodes are often simple and may experience significant drift. Moreover, the multi-hop character of wireless sensor networks complicates the problem, as one cannot simply employ a standard client/server clock synchronization algorithm as in wired networks.

As research in sensor networks evolved during the last years, many different approaches for clock synchronization were proposed [24, 25, 33]. *Reference Broadcast Synchronization (RBS)* [8] exploits the broadcast nature of the physical channel to synchronize a set of receivers with one another. A reference node is elected to synchronize all other nodes within a cluster. Since differences in radio propagation times can be neglected in sensor networks, a reference message arrives at the same instant at all receivers. The timestamp of the reception of a reference broadcast is recorded at each node and exchanged with other nodes to calculate relative clock offsets. Although being designed for single-hop time synchronization only, RBS can be utilized for multi-hop communication as component of other clock synchronization algorithms [15].

The *Timing-sync Protocol for Sensor Networks (TPSN)* [10] elects a root node and builds a spanning tree of the network during an initial level discovery phase. In the synchronization phase of the algorithm, nodes synchronize to their parent in the tree by a two-way message exchange. Using the timestamps embedded in the synchronization messages, the child node is able to estimate the transmission delay and

the relative clock offset. MAC layer time-stamping is used to reduce possible sources of uncertainty in the message delay. However, TPSN does not use drift compensation which makes frequent re-synchronization mandatory.

This shortcoming is tackled by the *Flooding-Time Synchronization Protocol (FTSP)* [19]. FTSP elects a root node based on the smallest node identifier and forms an ad-hoc tree structure by flooding the current time information of the root node into the network. Each node uses a linear regression table to convert between the local hardware clock and the clock of the reference node.

Although FTSP provides good global synchronization at low communication cost on small networks, it potentially incurs large skews between neighboring nodes due to the large stretch of the employed tree structure. This problem is avoided by the *Gradient Time Synchronization Protocol (GTSP)* [29], which follows a completely distributed scheme where nodes synchronize to all of their neighbors. While ensuring global synchronization comparable to FTSP even for small networks, skew between neighboring nodes is significantly smaller.

Rapid Time Synchronization (RATS) [16] employs a network-wide broadcast to disseminate the local time of the root node. As in the FTSP protocol, linear regression is used to estimate and compensate clock skews.

Another approach to exploit all communication links has been proposed in [28], where the algorithm seeks to minimize the sum over all edges of the squared skews. This can be accomplished in a distributed manner by applying a gradient descent on the corresponding linear system. Similar to most of the practicable protocols proposed so far, a low message frequency is paid for by slow information dissemination, imposing skews at least linear in the network diameter.

The fundamental problem of clock synchronization has been studied extensively and many theoretical results have been published which give bounds for the clock skew and communication costs [18, 23]. In [4] a worst-case lower bound of $\Omega(D)$ on the system-wide synchronization achievable was shown, where D denotes the network diameter. Given the hardware clock drift, the algorithm presented in [30] minimizes the global skew.

Astonishingly, the worst-case skew between neighboring nodes is not independent of the network diameter. A lower bound of $\Omega(d+\log D/\log\log D)$ has been proved [9] for nodes in distance d of each other, which also holds if delay uncertainties are neglected and an adversary can decide when messages are sent [20]. This result has recently been strengthened to $\Omega(d+\log D)$ and matched up to small constants [17].

However, we argue that these well-established theoretical lower bounds are based on too harsh assumptions to describe the practical limitations of clock synchronization. Typically they are proved by taking adversarial control of message delay jitters and/or clock drifts. However, jitters are random in nature and clock drift does not vary arbitrarily, at least not arbitrarily quickly. We believe that this approach covers a quite general class of distributed systems. Apart from sensor networks, our results may apply to acoustic networks (see e.g. [31]), for instance.

3 Model

We model a (sensor) network of $|V| =: n$ nodes as undirected graph $G = (V,E)$ of diameter D, where edges $\{v,w\} \in E$ represent bidirectional communication links. A message sent by a node $v \in V$ is received by all of its neighbors $w \in N_v$. For the purpose of our analysis, we assume that communication is reliable and the network is static. However, it takes messages some time to arrive at their destination. More precisely, the *message delay* is the time between a node deciding to send a message and the receiver finally processing it. This delay is in the order of milliseconds in wireless sensor networks. For several reasons, such as accuracy of computation, clock granularity, and scheduling of local operations, the message delay is not determined exactly, i.e., the system suffers from random deviations of the message delay, the so called *message delay jitter* [10]. By means of MAC layer timestamping, the jitter can be kept close to the limits given by the clock granularity [19]. In our model, we assume that remaining jitter in the message delay is normally distributed with a standard deviation of $\mathcal{J}$ and mean $\mathcal{T}$ or is uniformly distributed in $[\mathcal{T}-\mathcal{J},\mathcal{T}+\mathcal{J}]$.

Each node v is equipped with a *hardware clock* $H_v(t)$ whose value at real time t is

$$H_v(t) = H_v(0) + \left\lfloor \int_0^t h_v(\tau)\, d\tau \right\rfloor$$

where $H_v(0)$ is the *hardware offset* and $h_v(\tau)$ is the *hardware clock rate* of node v at time τ. The rate of the hardware clock depends on conditions like the temperature, the supply voltage, or crystal quartz aging, all changing over time, but even under perfectly stable and identical environmental conditions the clocks of different nodes will run at different speeds. However, we assume that

$$\forall v \in V\ \forall t : |1-h_v(t)| \leq \rho \ll 1,$$

i.e., the clocks exhibit a bounded drift. Typical hardware clock drifts are around 30 parts per million (ppm), while the fluctuations due to temperature are substantially smaller.

On top of the hardware clocks, nodes compute *logical clock* values $L_v(t)$. The goal of a *clock synchronization algorithm* is to ensure that these values are as closely synchronized as possible both between any two nodes in the network and between neighbors. This is measured by the (maximum) *global skew*

$$\mathcal{G}(t) := \max_{v,w\in V}\{|L_v(t)-L_w(t)|\}$$

and the (maximum) *local skew*

$$\mathcal{L}(t) := \max_{v\in V, w\in N_v}\{|L_v(t)-L_w(t)|\}.$$

Correspondingly, the average global and local skews are the average skews between all pairs of nodes respectively neighbors.

Certainly, these objectives can only be accomplished if nodes exchange messages about their state on a regular basis. Nevertheless, an algorithm should strive to minimize the communication necessary to maintain synchronization. Although we do not impose any explicit constraints on the size

of messages in our model, they are expected to be small, typically containing only a few bytes, e.g., a clock value and one or two variables.

4 Analysis

We now derive a lower bound on the global skew that a quite general class of algorithms in any system satisfying our model will experience. Afterwards we will examine why current synchronization algorithms do not achieve matching synchronization quality. To this end, we exemplarily show that both FTSP and GTSP exhibit significantly larger global skews than necessary. Our observations will give rise to the PulseSync algorithm proposed in Section 5.

4.1 Lower Bound on the Global Skew

Synchronizing the clocks of different sensor nodes involves the exchange of timing information over the radio channel. Several protocols for time synchronization use a reference or root node which disseminates its current clock value every beacon interval B. Therefore, an important prerequisite for every clock synchronization algorithm is the ability to estimate the rate at which the clock of another node advances. Hence, we examine how well the relative hardware clock rates can be estimated by the system even if they remain constant within an interval of length kB, $k \in \mathbb{N}$, and nodes base their estimate on k clock values.

LEMMA 4.1. *Assume that the hardware clock rates h_v and h_w of neighbors $v, w \in V$ are constant, and v sends $k \geq 2$ messages to w within a time interval of length kB, which do not depend on events that happened before that interval. If the jitter of the messages is normally distributed with a standard deviation of $\mathcal{J}$ or uniformly distributed in $[-\mathcal{J}, \mathcal{J}]$, the probability that any estimate $\hat{r}_w^v$ of the relative hardware clock rate $r_w^v := h_v/h_w$ that w computes has an error of at least $\Omega\left(\mathcal{J}/\left(Bk^{3/2}\right)\right)$ is constant, i.e.,*

$$P\left[|\hat{r}_w^v - r_w^v| \geq \Omega\left(\frac{\mathcal{J}}{Bk^{3/2}}\right)\right] \geq \frac{1}{2}.$$

PROOF. It takes $\mathcal{T} \pm \mathcal{J}$ time to send a message, i.e., the hardware clock values w received from v are affected by a random jitter with standard deviation $\mathcal{J}$. Note that any values apart from v's hardware clock value at the time of sending are of no use to w, since they are afflicted by the same error.[1]

Therefore, dividing the difference of two such values by the hardware time of w that has passed between the two messages containing them, an estimate of r_w^v with standard deviation in the order of $\mathcal{J}/(Bk)$ is obtained. This way, at most $k-1$ independent estimates of r_w^v can be computed, all suffering from a statistical error in the order of $\mathcal{J}/(Bk)$. Observe that r_w^v is the mean of this distribution. Since by linearity of expectation variance is additive, the sum of $k-1$ independent estimations of r_w^v has standard deviation $\Theta(\mathcal{J}/(B\sqrt{k}))$, implying that we have a constant probability that an error in this order occurs.[2] Thus, with probability $1/2$, any approximation of r_w^v based on this data sustains an error of $\Omega\left(\mathcal{J}/\left(Bk^{3/2}\right)\right)$. □

Note that this result holds for virtually any reasonable distribution of the jitter. Though this lemma seems to suggest to measure clock rates based on time intervals as large as possible, apparently this approach fails if clock rates change significantly within the observation period. A global bound can easily be followed from this local statement.

COROLLARY 4.2. *Suppose the jitter is normally distributed with a standard deviation of $\mathcal{J}$ or uniformly distributed within $[-\mathcal{J}, \mathcal{J}]$. Assume that hardware clock rates are constant, and a node w computes an estimate $\hat{r}_w^v$ of the relative hardware clock rate $r_w^v := h_v/h_w$ compared to a node v in distance d hops. If this estimate is based on k messages from v that have been forwarded to w without replication (i.e., following a single path and intermediate nodes send also only k messages) and are sampled from a time interval of length kB, with constant probability the estimate has an error in the order of $\mathcal{J}\sqrt{d}/\left(Bk^{3/2}\right)$, i.e.,*

$$P\left[|\hat{r}_w^v - r_w^v| \geq \Omega\left(\frac{\mathcal{J}\sqrt{d}}{Bk^{3/2}}\right)\right] \geq \frac{1}{2}.$$

PROOF. The "noise" induced by the jitter in the message delay accumulates on the path from v to w. Since we assume that the clock values are forwarded without replication, w can reduce this effect by no means. As the jitters of different messages are independent, the standard deviation of their sum grows as $\mathcal{J}\sqrt{d}$ with respect to the distance d. Now the proof proceeds analogously to the one of Lemma 4.1. □

In theory, replicating messages can improve the situation in that the influence of the noise can be reduced. However, intermediate nodes sending the same information twice is at odds with the goal of achieving a low message complexity for *all* nodes—and if more messages are sent by some nodes anyway, we might likewise increase the number of transferred clock values. On the other hand, in some graphs different paths to a node may give multiple independently forwarded copies of the same clock value "for free". Anyhow, in general we cannot expect multiple paths between two nodes, and even less between *any* two distant nodes. Furthermore, exploiting specific topologies may become quite cumbersome, especially so in case of dynamic systems.

Having derived the required preliminary statements, we now can conclude our lower bound on the global skew.

THEOREM 4.3. *Assume that $\rho \geq \frac{\mathcal{J}\sqrt{D}}{Bk^{3/2}}$ and the preliminaries of Corollary 4.2 are fulfilled. Using the same notation as in Corollary 4.2, within any time interval of length kB with constant probability a time t with a global skew of*

$$\mathcal{G}(t) \in \Omega\left(\frac{\mathcal{J}\sqrt{D}}{\sqrt{k}}\right)$$

exists.

PROOF. Since due to Corollary 4.2 relative clock rates cannot be measured more accurate than $\mathcal{J}\sqrt{D}/\left(Bk^{3/2}\right)$ between two nodes in distance of the network diameter D, the relative drifts of logical clocks are at least that large unless the maximum hardware clock drift ρ is already smaller. Thus, in order

[1] If messages of v are triggered by messages of w or other nodes, the error gets even larger.

[2] $\mathrm{Var}(X) = E\left(X^2\right) - E(X)^2$, and in case of identical variance of each summand, giving equal weights to all single estimations minimizes the variance of the final outcome.

to achieve better synchronization than $\Omega(\mathcal{J}\sqrt{D}/\sqrt{k})$, one of the nodes needs to estimate a clock value of the other which is not older than $o(T)$ time as accurate as $o(\mathcal{J}\sqrt{D}/\sqrt{k})$. However, we already observed that this would require $\omega(k)$ messages, contradicting the assumption that estimates are based on at most k values. □

At first glance this bound seems quite weak, as it gets arbitrarily small for large k. Moreover, even better rate approximations—permitting to send updates less frequently—could be computed using large observation intervals. However, this approach has its limitations, as this in turn enforces a slow adaption to dynamic changes of any kind, such like changes in topology or clock rates.

4.2 Weaknesses of Current Protocols

We analyze the behavior of two time synchronization protocols, FTSP and GTSP, which are specially tailored for wireless sensor networks. We do not focus on bootstrap problems, package loss or the impact of dynamic network topologies in our analysis.

4.2.1 Flooding Time Synchronization Protocol

After the initialization phase, FTSP utilizes a spanning tree rooted at the root node r to disseminate information on r's state throughout the network. Every node tries to synchronize its logical clock best possible to the one of r, while adjusting its clock value whenever an offset is detected. To this end, nodes broadcast their estimate of the reference clock value once every beacon period B. They store the k most recent values received from their parent in a table and determine their current estimate of r's clock value and the relative clock rate by means of a linear regression of this data set. Furthermore, when receiving a new message from their parent, they immediately update their logical clock rate and offset.

The decisive weakness of the protocol is the fact that errors introduced by jitter affect not only clock values, but also all clock *rates* further down the tree. Since the nodes send messages uncoordinatedly, information propagates merely one hop in $B/2$ time. In that time, the error is amplified by a constant factor! For ease of presentation, we restrict ourselves to the case of a regression table with only two entries ($k = 2$). Nevertheless, our simulations show that the same problem occurs in case of reasonable sizes of the regression table (see Figure 2). FTSP is simulated on a line topology, the root was fixed to be the first node of the line. The jitter in the message delay is chosen uniformly at random between -1 µs and 1 µs. Furthermore, there is no hardware clock drift, i.e., the errors introduced in the system are only due to the message delay jitter and the linear regression. The TinyOS implementation of FTSP ignores estimates which are seemingly invalid due to large skews; it even resets the regression table if this happens repeatedly (cf. Section 6). This has been suppressed in the simulations to demonstrate the behavior of the basic protocol for large diameters.

THEOREM 4.4. *Suppose that the jitter is normally distributed with a standard deviation of $\mathcal{J}$ or uniformly distributed within $[-\mathcal{J},\mathcal{J}]$. Then, even if the hardware clock rates are constant, at any time t there is a constant probability that the FTSP protocol with parameter $k = 2$ exhibits a global skew*

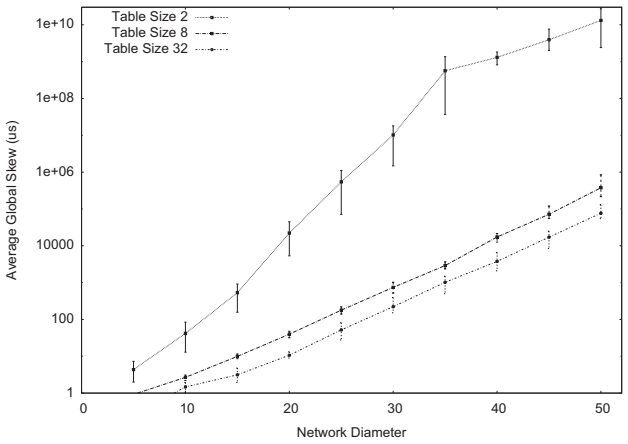


Figure 2: Simulation of FTSP for line topologies with different diameters using varying sizes $k = 2, 8, 32$ of the regression table. Mean synchronization errors averaged over five runs, error bars indicate values of runs with maximum and minimum outcome.

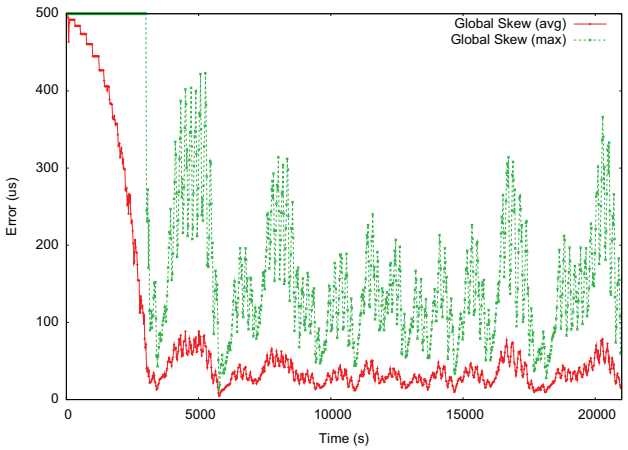


Figure 3: Simulation of FTSP on a line topology of 20 nodes. Average and maximum network error plotted against time. Regression table size 8, jitter uniformly random ± 0.25 μs, clock drift ± 10 ppm, first node is root.

exponential in the network diameter D, i.e.,

$$P\left[\mathcal{G}(t) \geq \mathcal{J}2^{\Omega(D)}\right] \geq \frac{1}{2}.$$

PROOF. For simplicity, assume that the diameter D is even and no node sends two messages in a time interval where its neighbors do not send a message. Then, any estimate of L_r received by some node v is kept for an expected time of $B/2$ (since sending times are not aligned) and then the current estimate is forwarded before the next value is received. Since the diameter of the graph is even, a node $v_{D/2}$ exists receiving estimates of L_r that are forwarded along a path $v_1, \ldots, v_{D/2}$.

Now assume that everything is perfect until a time t_0 where r sends the message m: All logical clock values and

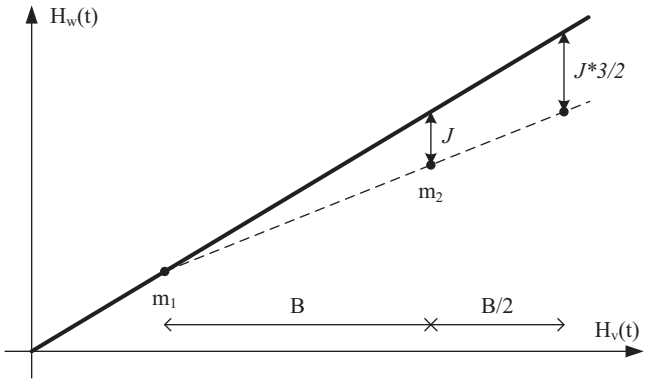


Figure 4: FTSP logical clock computation scheme for the special case $k = 2$. Errors introduced by jitter are amplified because they affect both the axis intercept and the gradient of the regression line.

rates are equal to the ones of r. The message m contains the clock value $L_r(t_0)$, which is received by v_1 at time t_1. Node v_1 computes an estimate of $L_r(t_1)$ which is accurate up to the error introduced by the jitter $\mathcal{J}$.

However, apart from the additive error of the estimate (and thus v_1's logical clock, which in case of $k = 2$ is simply set to the received value), now the logical clock *rate* l_1 will differ from $l_r = h_r$. This error has standard deviation $\mathcal{J}/B$, since the (relative) rate of v_1 is simply computed as the difference of the last two values received from r divided by the (local) time passed in between. After expected time $B/2$, v_1 forwards its estimate, implying that the error of the clock rate induces an expected additional error of roughly $\mathcal{J}/2$ in the estimate received by v_2 (see Figure 4 for illustration). The crucial point is that this error is not random, but amplifies the jitter on the message received by v_1! Thus, node v_1 forwards the same error, but (in expectation) multiplied by $3/2$. By induction, the same is true for all v_i, $i \in \{2,\dots,D/2\}$. We conclude that the estimate of L_r received by v_D suffers from an error exponential in D, as the situation certainly does not improve if we do not assume perfect initial conditions or take the additional errors due to the jitter on different edges into account. □

4.2.2 Gradient Time Synchronization Protocol

The GTSP protocol does not exhibit this exceptionally bad growth of the global skew. The basic idea of the algorithm is that logical clock rates are determined by perpetually measuring the relative clock rates of neighboring nodes and setting the own rate to the average of the outcomes of these measurements. An important feature of this technique is that rates are independent of observed logical clock skews, therefore errors are not amplified as by the FTSP protocol.

However, this imposes the need to ensure that small skews do not accumulate over time. Hence, nodes regularly increase their logical clock values by the average skew to neighboring clocks. For a network diameter $D = 10$, this results in a global skew comparable to the one of FTSP, but averaging over all neighbors yields a much smaller local skew.

The employed message pattern is identical to FTSP and is the reason why GTSP and similar algorithms fail to achieve a global skew of $O(\mathcal{J}D)$.

THEOREM 4.5. *Assume that GTSP synchronizes the logical clock rates l_v and l_w of two nodes v,w in distance d as good as $\mathcal{J}d^\alpha/B$, $\alpha > 0$, then GTSP exhibits at any time t with constant probability a global skew of $\Omega\left(\mathcal{J}D^{1+\alpha}\right)$.*

PROOF. Fix two nodes v,w at distance D from each other whose logical clock rates differ by $\mathcal{J}D^\alpha/B$ at some time. Any node u at distance $d \le D/4^{1/\alpha}$ will have a clock rate differing by no more than $\mathcal{J}D^\alpha/(4B)$ from v's, and likewise for nodes close to w. Therefore, all nodes within distance $D/4^{1/\alpha}$ from v have by $\mathcal{J}D^\alpha/(2B)$ faster (or slower) clocks than the nodes within distance from w.

Since it takes in expectation $DB/\left(2\cdot 4^{1/\alpha}\right)$ time until the state of nodes further away than $D/4^{1/\alpha}$ influence v's or w's clock rates at all and clock rates are determined by averaging, v's and w' clocks will (with constant probability) drift apart at the rate of at least $\mathcal{J}D^\alpha/(2B)$ for that time period. Thus, a skew of $\mathcal{J}D^{1+\alpha}/\left(2\cdot 4^{1/\alpha}\right) \ge \Omega\left(\mathcal{J}D^{1+\alpha}\right)$ is built up. Now, as the algorithm handles clock rates independently from logical clock skews, in one of the symmetric situations where either v's or w's logical clock runs faster, a global skew of $\Omega\left(\mathcal{J}D^{1+\alpha}\right)$ must occur. □

We give no proof here that the assumption of the theorem that the clock rates between nodes at distance d are synchronized like $\mathcal{J}d^\alpha/B$ holds. In fact, this may depend on the topology. However, it is reasonable to assume that close nodes' rates are synchronized better because the algorithm seeks to locally minimize their differences.

Another important observation following from this result is that many other reasonable algorithms (e.g. like the one in [28]) will experience a global skew that is (super)linear with respect to the diameter: Whenever clock rates are locally balanced and do only change dependent on neighbors' rates, they will not change faster than information propagates through the network. If sending times are not aligned, this will leave time for errors in clock rates to accumulate skews at least linear in the diameter.

5 The PulseSync Algorithm

We have observed two major issues that may degrade the quality of (global) synchronization: Firstly, increasing received estimates at rates computed with their help leads to self-amplification of errors. Secondly, if sending times are not aligned, information propagates slowly, giving time to pile up skew originating from small differences in clock rates. Both of these problems are already present if the system is static. Certainly, to cope with dynamics such as changing hardware clock rates or network topologies, quick dissemination of information is also a highly desirable property of an algorithm.

Hence, in this section we will devise an algorithm not suffering from these drawbacks and then prove a bound on its global skew.

5.1 Description of the Algorithm

For the purpose of our analysis, we will give a high-level description of the algorithm in pseudocode. In Section 6 the details of the implementation will be discussed.

The basic idea of the algorithm is to distribute information on clock values as fast as possible, while minimizing the number of messages required to do so. In particular, we want nodes to send messages only once in B time, while avoiding that it takes in expectation $\frac{B \cdot D}{2}$ time until distant nodes affect each other. Obviously, a node cannot forward any information it has not received yet, enforcing that information flow is directed: An intermediate node on a line topology has to wait for at least one message from a neighbor, while after its reception it ought to send as quickly as possible in order to not slow things down. Thus, we naturally end up with flooding a *"pulse"* through the network, implicitly building a breadth-first search tree. This technique further entails that pulses are generated by some node, which hence becomes the root r of the tree. This node is the only one which all nodes in the network obtain recent information from in a pulse, making its clock the canonical target of synchronization.

To keep clock skews small at all times, nodes employ a drift compensation, also relative to r. To even out the effects of the random jitter, this estimate is based on k values. This can be done in several ways; we choose a linear regression as also utilized by FTSP.

The problem with this scheme is that it obviously provokes lots of collisions. Therefore, a practical implementation will have to schedule the flooding. We discuss this issue in more detail in Section 7. In the abstract description of the algorithm, we represent this by nodes having to wait until they can send a message. This, however, raises the question how the estimate of the root's clock value L_r has to be adapted in the meantime. Even if messages could be relayed immediately after reception, we still had to compensate for the message delay $\mathcal{T}$. Increasing received estimates at the speed of the own logical clock (which also incorporates this estimate!) again evokes self-amplification of errors as in FTSP.[3] On the other hand, relying on the bare hardware clock rates may result in errors comparable to (or even dominating) the jitter. However, as the time between reception and forwarding of an estimate will be small compared to B, virtually any estimate $\hat{r}_v^r$ a node v acquires of the relative clock rate to r will do, as long as we avoid the self-amplification issue. Therefore, nodes compute an approximation of r_v^r by means of the first k messages they receive, while relying on their hardware clock (i.e., estimating relative rate 1) when forwarding the first.[4] Thus, we neither slow down the initialization process nor do we sacrifice accuracy if the topology is stable. The delay $\mathcal{T}$ can easily be incorporated by adding $\hat{r}_v^r(H_v(t_r) - H_v(t_r - \mathcal{T}))$ to a clock value received at time t_r, as within $\mathcal{T}$ time the root's clock makes a progress of $r_v^r(H_v(t_r) - H_v(t_r - \mathcal{T}))$.[5]

[3] Yet to a smaller extent, as nodes forward estimates as quickly as possible. Thus the exponential behavior would become evident at larger diameters than in case of FTSP.

[4] A similar approach has been used for high latency networks where the drift during message transfer is a major error source [31].

[5] Note that referring to $H_v(t_r - \mathcal{T})$ is an abuse of notation, since nodes are only able to access the current hardware time. This is to be understood as the timestamp v made at the time $t_r - \mathcal{T}$ when the transmission was initiated, where for simplicity we assume here that the sender is able to directly incorporate the corresponding clock value into the message.

The pseudocode of the algorithm for non-root nodes is given in Algorithm 2, whereas the root follows Algorithm 1. In the abstract setting, a message needs only to contain an estimate $\hat{L}_r$ of the root's clock value. We also add a sequence number i initialized to 0, but in fact it is only used to distinguish between two pulses, which in case of $B \gg \mathcal{T}$ is also easily possible without such a number. In practice, a message may contain additional useful information, such as an identifier, the identifier of the (current) root of a node, or the (current) depth of a node in its tree. For the root node, the logical clock is simply identical to the hardware clock. Any other node computes $L_v(t)$ as the linear regression of the k stored pairs of hardware clock values and corresponding estimates, evaluated at x-value $H_v(t)$.

Algorithm 1 Whenever $H_r(t) \bmod B = 0$ at the root node r.

1: send $\langle H_r(t), i \rangle$
2: $i := i + 1$

Algorithm 2 Node v receives first message $\langle \hat{L}_r, i \rangle$ with sequence number i at local time $H_v(t)$

1: store $\langle \hat{L}_r + \hat{r}_v^r(H_v(t) - H_v(t - \mathcal{T})), H_v(t), i \rangle$
2: delete $\langle \cdot, \cdot, i - k \rangle$
3: wait until time t' when allowed to send
4: send $\langle \hat{L}_r + \hat{r}_v^r(H_v(t') - H_v(t - \mathcal{T})), i \rangle$
5: $i := i + 1$

5.2 Upper Bound on the Global Skew

After devising PulseSync, we now prove a strong probabilistic upper bound on its global skew. To this end, we will first derive a bound on the accuracy of the estimates of relative hardware clock rates the nodes compute. Then we will proceed to bound the clock skews themselves.

We will need a statistical tool which is similar to Chernoff's bound. It basically states that deviations from the expected value of a random variable that are large compared to the square root of the expectation are highly unlikely.

LEMMA 5.1 (HOEFFDING'S INEQUALITY). *Given independent random variables $X_i \in [a_i, b_i]$, $i \in \{1, \dots, l\}$, for their sum $X := \sum_{i=1}^{l} X_i$ the inequality*

$$P[|X - E[X]| \geq l\delta] \leq 2e^{-2l^2\delta^2 / \sum_{i=1}^{l}(b_i - a_i)^2}$$

holds for any $\delta > 0$.

PROOF. See [12]. □

Next, we tailor this bound to our specific needs. The obtained result will be the basis for our reasoning on the global skew of PulseSync.

LEMMA 5.2. *Assume that X_i, $i \in \{1, \dots, kD\}$, $k \in 2\mathbb{N}$, are independent random variables with $X_i \in [C_i - \mathcal{J}, C_i + \mathcal{J}]$ and $E[X_i] = E$. Define $X := \frac{4}{k^2(1-\varepsilon)B} \sum_{i=1}^{kD/2} (X_{i+kD/2} - X_i)$ for some $\varepsilon \in (0,1)$. Then for any choice of $c > 0$ we have*

$$P\left[|X - E[X]| \geq \frac{2\mathcal{J}\sqrt{cD \ln n}}{k^{3/2}(1-\varepsilon)B}\right] \leq 2e^{-c \ln n} = \frac{2}{n^c}.$$

PROOF. We apply Hoeffding's inequality with $l = kD/2$ and $\delta = \mathcal{J}2c\ln n/(kD)$ to $(k/2)^2(1-\varepsilon)BX$. □

This result permits to bound the accuracy of the estimates $\hat{r}_v^r$ the nodes determine in the initialization phase.

LEMMA 5.3. *Suppose the jitter is random from an interval of length $\mathcal{J}$. Assume that if node $v \in V$ receives a message, it has to wait $\tau_v + \tilde{\tau}$ time until it can send itself, where τ_v is constant and $\tilde{\tau}$ is random with zero expectation from an interval of length at most $\mathcal{J}/\rho$.[6] If hardware clock rates are constant, the initial guess $\hat{r}_v^r$ a node $v \in V$ computes based on the first k received values carries with probability at least $1 - 2/n^c$ an error of at most*

$$|\hat{r}_v^r - r_v^r| \leq O\left(\frac{\mathcal{J}\sqrt{cD\ln n}}{k^{3/2}B}\right) =: \Delta_r,$$

for arbitrary $c > 0$.

PROOF. Since clock rates are constant, for any node v, L_r is a linear function of H_v. However, as v computes $\hat{r}_v^r$ by means of estimates of L_r, it uses a disturbed data set consisting of k pairs $\left(H_v(t_i), \hat{L}_r(t_i)\right)$, $i \in \{1,\ldots,k\}$, to approximate its relative logical clock rate. Any errors which are independent of i, such as the error induced by the clock drift of relaying nodes w during the τ_w expected time they wait for sending a message, cancel out when estimating the rate, i.e., $E[\hat{r}_v^r - r_v^r] = 0$. Thus, only the possible fluctuations of the jitter, bounded by $\pm\mathcal{J}/2$, and of $\tilde{\tau}h_w$ at relaying nodes, also bounded by $\pm\mathcal{J}/2$ due to the assumption that $\tilde{\tau} \leq \mathcal{J}/\rho$, affect $\hat{r}_v^r$.

Taking into account these observations, each of the values $\hat{L}_r(t_i)$ is disturbed by an independent random variable, which itself is the sum of at most D independent random variables $X_{i,j} \in [C_j - \mathcal{J}, C_j + \mathcal{J}]$. Now, a linear regression minimizes the least squares error (i.e., the "variance" of the data set compared to the computed line), yielding the maximum likelihood estimation. In other words, it is at least as good as any scheme, in particular the following one. Assume without loss of generality that $k \in 2\mathbb{N}$ (otherwise the center value is dropped). Compute m estimates of $\hat{r}_v^r$ as $\left(\hat{L}_r(t_{i+k/2}) - \hat{L}_r(t_i)\right) / \left(H_v(t_{i+k/2}) - H_v(t_i)\right)$, $i \in \{1,\ldots,k/2\}$, and afterwards take the average of the $k/2$ obtained values. The smaller the time interval between messages, the worse each single estimate. However, certainly these differences are lower bounded by $k(1-\varepsilon)B/2$ for some $\varepsilon \ll 1$, since hardware clocks progress roughly at the rate of real time. Therefore, inserting this bound, the error of the scheme is dominated by the absolute value of the random variable

$$X := \frac{4}{k^2(1-\varepsilon)B}\sum_{i=1}^{m}\sum_{j=1}^{D}(X_{i+k/2,j} - X_{i,j})$$

with expectation $E[X] = 0$. By an appropriate reordering of the indexes, we see that Lemma 5.2 can be applied to this variable, which completes the proof. □

Obviously, the error of the initially computed estimate might get larger if it is dominated by variations in hardware clock rates. Nevertheless, even under extreme environmental conditions (e.g., variations in temperature of 50 degrees), in practical systems this error will be smaller than the initial differences of the hardware clock rates.

Based on the preceding observations, we now can prove a bound on the skew between a node and the root.

THEOREM 5.4. *Assume that the preliminaries of Lemma 5.3 are fulfilled and $\Delta_r \leq \rho$.[7] Denote by $v_0 := r, v_1, \ldots, v_d := v \in V$ the path from the root to v along which estimates of r's clock values are forwarded. Set $p_v := \sum_{i=1}^{d}(\tau_{v_i} + \mathcal{T})$, i.e., the expected time an estimate "travels" along the path. Then the probability that for all times $t \in [t_1, t_2)$ between v receiving two messages, where at least $2k$ pulses are complete, the inequality*

$$|L_v(t) - L_r(t)| \leq O\left(\mathcal{J}\sqrt{\frac{cD\ln n}{k}}\left(1 + \frac{p_v}{kB}\right)\right)$$

holds, is at least $1 - 1/n^c$.

PROOF. Since at least $2k$ pulses are complete, all estimates are locally increased at the rates $h_w\hat{r}_w^r$ which differ with probability at least $1 - 2/n^c$ by no more than Δ_r from h_r. The latest k values $\hat{L}_r(t_i)$, $i \in \{1,\ldots,k\}$ that v (and all intermediate nodes) received are independent of the first k. Therefore, using the same argumentation as in Lemma 5.3 and the assumption that $\Delta_r \leq \rho$, the logical clock rates of all v_i, $i \in \{1,\ldots,d\}$, differ by at most Δ_r from h_r with probability at least $1 - 2d/n^c$.

The estimates received by v will suffer from an error with expected absolute value of at most $\Delta_r p_v$.[8] Hence, these errors are dominated by independent random variables $X_i := \Delta_r p_v + |\sum_{j=1}^{d} X_{i,j}|$, $j \in \{1,\ldots,d\}$, where $X_{i,j}$ are the independent random errors made at each hop, i.e., $X_{i,j} \in [-\mathcal{J}, \mathcal{J}]$. The logical clock value of v at time t is computed as $\bar{L}_r + l_v(t - \bar{t})$, where l_v is the logical rate of v (i.e., h_v times the gradient of the regression line), $\bar{t}$ is the average of the times where the last k estimates were received, and $\bar{L}_r$ is the average of these estimates. We have that $t - \bar{t} = \Theta(kB)$ and $|l_v - h_v| \leq \Delta_r$.

Therefore, we need to bound $|\bar{L}_r - L_r(\bar{t})|$. Since h_r is constant and $\bar{t}$ is the mean of the times the estimates were received, we simply have that

$$\begin{aligned}|\bar{L}_r - L_r(\bar{t})| &= \frac{1}{k}\left|\sum \hat{L}_r(t_i) - L_r(t_i)\right| \\ &\leq \frac{1}{k}\sum_{i=1}^{k} X_i = \Delta_r p_v + \frac{1}{k}\left|\sum_{i=1}^{k}\sum_{j=1}^{d} X_{i,j}\right|.\end{aligned}$$

We estimate $d \leq D$ and apply Hoeffding's inequality with $l = kD$ and $\delta = \mathcal{J}\sqrt{c\ln n/(2kD)}$ to obtain that with probability at least $1 - 2/n^c$, we have that

$$|\bar{L}_r - L_r(\bar{t})| \leq O\left(\Delta_r p_v + \mathcal{J}\sqrt{\frac{cD\ln n}{k}}\right).$$

We conclude that with probability at least $1 - (4+D)/n^c$ it

[6] Note that $\mathcal{J}/\rho$ is typically in the order of 10^{-1} or more, i.e., in practice this assumption is no restriction.

[7] Examining Δ_r reveals that this inequality certainly holds for practical values.

[8] Note that due to lack of independence of the estimated relative rates $\hat{r}_w^r$, we cannot guarantee anything better than linear accumulation of errors here.

holds that

$$\begin{aligned} |L_v(t) - L_r(t)| &\leq |(l_v - h_r)|\,(t-\bar{t}) + |\bar{L}_r - L_r(\bar{t})| \\ &\leq O\left(\Delta_r kB + \Delta_r p_v + \mathcal{J}\sqrt{\frac{cD\ln n}{k}}\right) \\ &\leq O\left(\mathcal{J}\sqrt{\frac{cD\ln n}{k}}\left(1+\frac{p_v}{kB}\right)\right). \end{aligned}$$

Finally, since c was arbitrary, we may simply replace c by a slightly larger c' such that the inequality holds with probability at least $1-(4+D)/n^{c'} \geq 1-1/n^c$, which concludes the proof. □

This powerful statement on the probability of large errors leads to the desired bound on the global skew.

COROLLARY 5.5. *Let t_i denote the time where the* i*th pulse is complete. Provided that the prerequisites of Theorem 5.4 are satisfied, the total number of pulses P is polynomial in n, and* $\max_{v\in V}\{p_v\}/(kB) \leq O(1)$*, we have that*

$$P\left[\max_{t\in[t_{2k},t_P]}\{\mathcal{G}(t)\} \leq O\left(\mathcal{J}\sqrt{\frac{D\ln n}{k}}\right)\right] \geq 1-\frac{1}{n}.$$

PROOF. Appropriate choice of c in Theorem 5.4. □

Taking all estimations made into account, this means for a line topology featuring 20 nodes, $k=8$, and $\mathcal{J}=\pm 1$, that the global skew will with a probability of at least 95% never exceed roughly 12 microseconds within 1000 pulses (excluding initialization). Certainly, the average skew will be significantly smaller. These results are in excellent accordance with the results of the simulations (see Figure 5).

Considering that the $\sqrt{\ln n}$ factor takes the role of reducing the probability of error from a constant to a negative power of n, the given upper bound matches the lower bound given in Theorem 4.3. Strictly speaking, the requirement that the time $\max_{v\in V}\{p_v\}$ a pulse takes, which is at least linear in D, is smaller than kB still implies that (asymptotically) the algorithm cannot adapt faster to dynamics than the obvious lower bound prescribed by the product of diameter and delay. For practical purposes, though, this is dominated by the time B between to pulses. This is another significant advantage compared to FTSP and any other synchronization algorithm employing a temporally not aligned message pattern.

6 Testbed Experiments

We present experimental results for PulseSync to verify that our theoretical results also apply in practice. To that end, we have implemented PulseSync on the Mica2 platform. Furthermore, we compare our approach to FTSP, the state-of-the art clock synchronization protocol for wireless sensor networks.

6.1 Hardware Platform

The hardware platform used for the implementation of PulseSync is the Mica2 sensor node [11]. The Mica2 platform features an Atmel ATmega128L microcontroller with 4 kB of RAM, 128 kB program ROM and 512 kB external flash storage. The TI CC1000 radio module offers data rates up to 76.8 kBaud using frequency shift keying (FSK). We use the 7.37 MHz quartz oscillator as the system clock source, and configure a 16-bit timer to operate at 1/8 of the oscillator frequency, resulting in a clock frequency of 921 kHz. An additional 16-bit variable keeps track of counter overflows, providing us in the end with a 32-bit hardware clock which offers a precision of roughly a microsecond.

6.2 PulseSync Implementation

The implementation of the PulseSync protocol on the Mica2 platform is done using TinyOS 2.1 [1]. The protocol implementation provides access to a synchronized global clock for other components of the application running on the node.

After startup, the *PulseSync* component is initialized by setting the logical clock to the current hardware clock value. By overhearing the channel for other synchronization messages, a node will quickly learn about the presence of a reference node. If no synchronization messages have been received for a certain time, the node will declare itself as the reference node and starts to broadcast periodic synchronization pulses.

A node will receive periodic pulses from nodes located closer to the reference node. If the reference node should suffer from a hardware failure or the link to our predecessor node is broken, no new sync pulses will be received. When no synchronization messages are received from a node for several consecutive beacon intervals, the node starts again to advertise itself as the reference node.

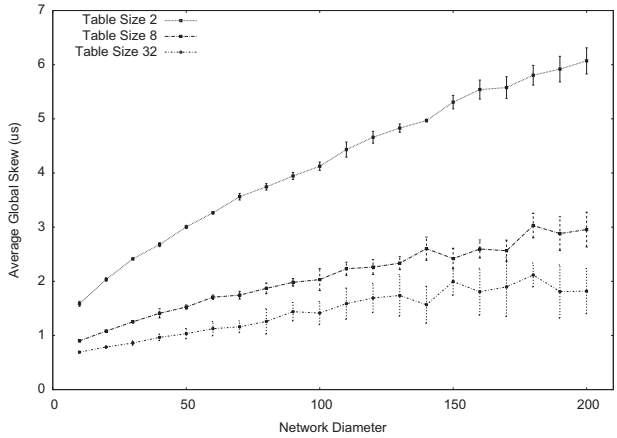
Depending on the network topology, a node will receive synchronization messages from multiple nodes in its neighborhood. Only the first pulse arriving at a node will be forwarded. It is very likely that this pulse has travelled on the shortest path from the root node and, therefore, the jitter introduced along the path is assumed to be smaller than on longer paths. To detect duplicate pulses, we insert a sequence number field into the synchronization message which is increased by the reference root after each sent pulse.

6.2.1 Synchronous Acknowledgments

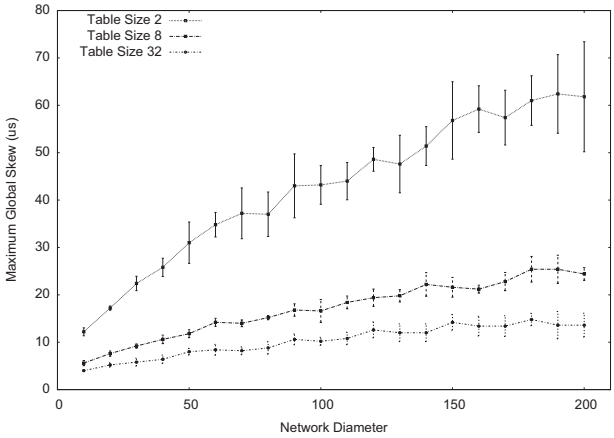
Clearly, the performance of any clock synchronization protocol is degraded when synchronization messages do not reach their destination due to a lossy communication channel. This is a problem each protocol has to cope with in a real-world deployment. However, in our testbed experiments we are interested in the actual performance of our clock synchronization protocols, PulseSync and FTSP. To mitigate the effects of packet loss, we implemented synchronous acknowledgments on the application layer for both protocol implementations.

6.2.2 Packet Timestamping

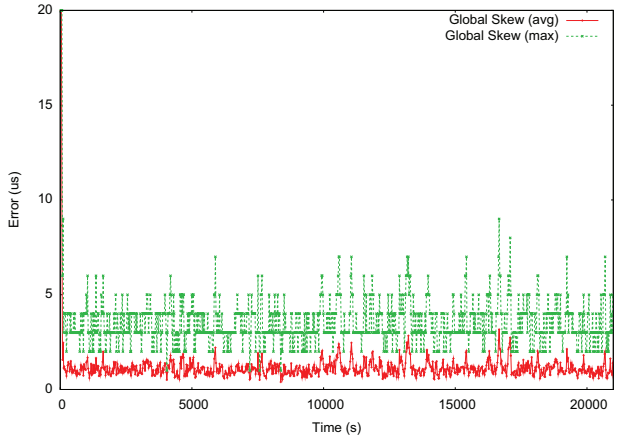
The time it takes to send a message containing a timestamp from one node to another is highly non-deterministic due to various error sources in the message path [10, 14]. It is well-known that time-stamping messages at the MAC layer can reduce most of this non-deterministic delays. An outgoing message is time-stamped just before the packet is transmitted over the radio channel. On the receiver side, the timestamp is recorded when the first byte of the message preamble arrives at the radio chip. The FTSP time-stamping scheme [19] records multiple timestamps at the byte boundaries to further reduce invariance due to delays in the inter-



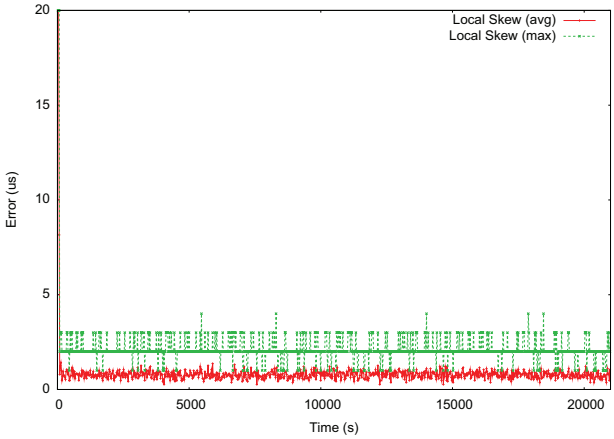
(a) Average Global Skew against Diameter



(b) Maximum Global Skew against Diameter



(c) Global Skew against Time (regression table size $k = 8$)



(d) Local Skew against Time (regression table size $k = 8$)

Figure 5: Simulations of the PulseSync algorithm on line topologies. Jitter is uniformly random ± 1 μs , clock drift ± 30 ppm, time between pulses is $B = 30$ s. The plots against diameter were obtained by averaging over 5 runs for each diameter, each being observed for 1000 pulses (excluding initialization). Nodes were allowed to send immediately when receiving a message, as the corresponding term in the estimation is small. The root was fixed to be the first node.

rupt handling involved in this approach. In our implementation of the MAC layer timestamping the first six bytes are time-stamped on both sides to reduce the jitter in the message delay.

6.3 Testbed Setup

We use an indoor testbed of 20 Mica2 sensor nodes which are placed in close proximity, thus forming a single broadcast domain. The actual network topology used in the experiments is enforced in software. A base station node connected to a PC is used to monitor the experiment. To test the synchronization accuracy of the two different clock synchronization protocols, we rely on external events triggered by the arrival time of special probing packets. On reception of such a time probe, each node writes both its hardware and logical time to the external flash memory. At the end of the experiment, the measurement results stored in the flash memory of the nodes are transferred to the PC for further analysis. The time interval between time probe events is uniformly distributed between 18 and 22 seconds. This guarantees that the time interval between the clock synchronization packets and the arrival of the time probes is changing continuously.

We measured the performance of PulseSync and FTSP on a line topology of 20 nodes. While many previous sensor network deployments did not exceed a network diameter of a very few hops, we believe that technical advances may lead to deployments with larger network diameters. Networks for continuous surveillance of sensor points located over large distances (e.g., roads, tunnels or pipelines) will typically form a line-like topology.

We used the latest FTSP implementation available from the TinyOS 2.x CVS repository during our experiments. Using the default parameter settings, FTSP failed to synchronize all nodes in the network to the reference node even after a long time period. We observed that it takes many

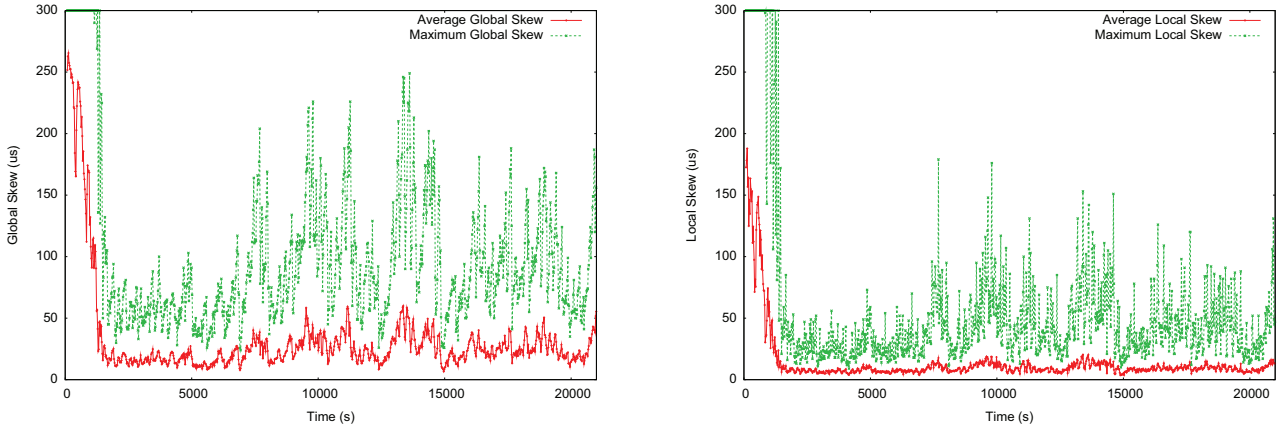


Figure 6: Global (left) and local skew (right) measured for FTSP on a line topology of 20 Mica2 sensor nodes. For times $t > 2000$ s, we observed an average global skew of 23.96 µs while the maximum error between any two nodes in the network was 249 µs. The local skew of FTSP was 9.04 µs on average and at most 179 µs.

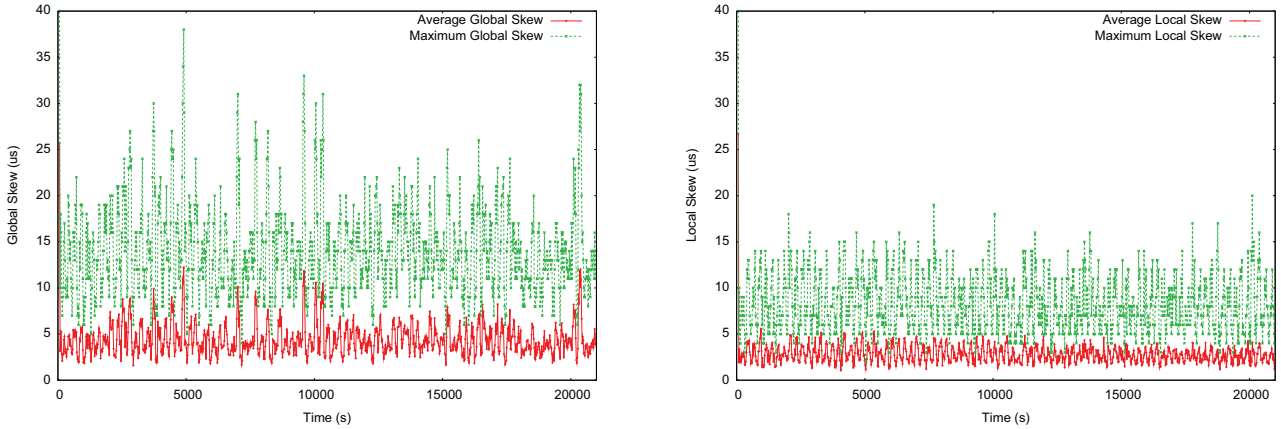


Figure 7: Global (left) and local skew (right) measured for PulseSync on a line topology of 20 Mica2 sensor nodes. For times $t > 100$ s, we observed an average global skew of 4.44 µs, the maximum error was 38 µs. The local skew was on average 2.79 µs and at most 20 µs.

rounds until all nodes agreed on the same reference node. After all nodes have been started, it takes `ROOT_TIMEOUT` rounds until multiple nodes in the network start to claim being the reference node by broadcasting synchronization beacons. Then, these new reference nodes will ignore beacons from other nodes having a lower identifier for the next `IGNORE_ROOT_MSG` rounds. If a node learns about a new reference node with a lower identifier, it will wait until it is synchronized well enough to this new reference node by listening to `ENTRY_SEND_LIMIT` beacons before it will forward beacons itself. Another issue with FTSP is the fact that nodes will clear the content of their regression table when the synchronization error between the received beacon and the estimated time of the node exceeds the `ENTRY_THROWOUT_LIMIT` parameter. Such a node will stop sending its own synchronization beacons until it has re-established synchronization to its reference node again. However, if the local skew incurred by the protocol gets large, this behaviour does not protect against infrequent outliers or failures, but contrariwise worsen the situation. The next node in the line will think that the current reference node died and it will start a new leader election round by advertising itself as the new root node. Since such effects will clearly degrade the performance of any clock synchronization protocol, we slightly modified FTSP to work with a fixed network topology (node 1 is the reference node).

Furthermore, we set `ENTRY_THROWOUT_LIMIT` to the maximum integer value to prevent nodes from clearing the regression table. For a fair comparison, we also enforced a fixed topology with Node 1 as the root for PulseSync.

6.4 Measurement Results

We collected measurement results during a 6 hour run for both PulseSync and FTSP. This resulted in roughly 20,000 data points collected from the sensor nodes. In our evaluation we focus on two different metrics: the *local skew* and the *global skew* (as defined in Section 3) are calculated for

each time probe event. We are particularly interested in the maximum global skew so that we can get an impression on the worst-case accuracy of both clock synchronization protocols.

Figure 6 shows the measurement results for FTSP on a line of 20 Mica2 nodes. It can be seen that it takes FTSP roughly 2000 seconds until all nodes are synchronized. Measurement results for the implementation of PulseSync are presented in Figure 7. PulseSync converges rapidly to a common logical time since the time information is flooded in a single round from the reference node to all other nodes throughout the network. Although PulseSync and FTSP have the same message complexity (one message per node and synchronization period), the fast flooding approach of PulseSync achieves a significantly improved synchronization accuracy on the same network topology. Furthermore, our experiments confirm our finding that the synchronization error to the root node must increase exponentially when using FTSP, while PulseSync performs significantly better (see Figure 8). The measurement results are summarized in Table 6.4. We can see that the synchronization error on the real sensor node hardware is consistent with what we expect from the simulations, although it seems that the simulations are based on too optimistic assumptions about clock drift and/or jitter. In particular, we assumed perfectly stable hardware clocks, but at the level of accuracy provided by PulseSync even small fluctuations in the clock rates become visible.[9]

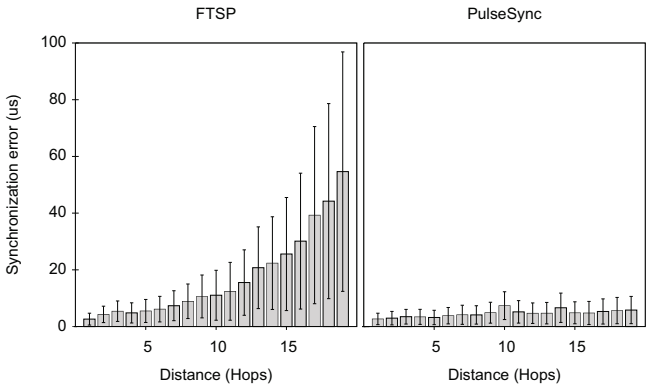


Figure 8: Synchronization error versus distance from the root node for FTSP (left) and PulseSync (right).

	FTSP	PulseSync
Synchronization messages	13,510	13,504
Global skew (avg)	23.96 µs	4.44 µs
Global skew (max)	249 µs	38 µs
Local skew (avg)	9.04 µs	2.79 µs
Local skew (max)	129 µs	20 µs

Table 1: Measurement results for FTSP and PulseSync on a line of 20 Mica2 sensor nodes.

[9]Even a single degree of temperature difference may lead to an absolute drift of roughly ± 10 µs within 30 seconds (cf. Figure 1).

7 Protocol Improvements

In this section we discuss two improvements to the basic PulseSync algorithm. First of all, an efficient broadcasting mechanism is vital for more complex and dense topologies. Then, we discuss some approaches to further reduce the local skew.

7.1 Wireless Scheduling

As we learned in this paper, time information must be propagated as quickly as possible. In a wired network, PulseSync can be implemented directly by a simple flooding algorithm. In wireless networks, scheduling the transmissions is more difficult because of interference. This issue has been studied arduously in the theory community; it is generally known as the broadcast problem in radio networks.

More than 20 years ago, it was shown that finding an optimal broadcast schedule is NP-hard [5]. One may approximate the optimal solution [6], however, at a communication cost which is clearly beyond anything practically tolerable. In addition, these early centralized broadcast solutions are designed for graphs, and graphs do not model wireless transmissions well [22].

More promising are distributed algorithms, on unknown topologies. Bar-Yehuda et al. [3] show that a straightforward randomized protocol will reach all nodes with high probability in $O(\mathcal{T}D\log^2 n)$ time. This result can even be improved slightly [7], matching a known lower bound.

One may improve the broadcast time even more by making the graph sparse by computing a connected dominating set [26] first. After receiving the PulseSync message, nodes in the connected dominating set simply transmit with constant probability, logarithmically often. Assuming that the interference radius is only a constant factor larger than the transmission radius, this broadcast algorithm finishes in $O(\mathcal{T}D\log n)$ time with high probability. In other words, even on a slow Mica2 radio all nodes can be reached in less than a second.

Although we did not employ these sophisticated algorithms in our prototype implementation, they demonstrate that efficient solutions of the problem are feasible. With such techniques, for a reasonable choice of B (30 seconds or more) it should be possible to complete pulses in less than B time on any real-world topology; even if this was not the case, a new pulse could be started before the previous one is finished. In B time the nodes that are active because of the current pulse will be far away from the root, hence this will not cause additional collisions. From Theorem 5.4 we see that in this scenario it suffices to complete pulses within kB time to maintain the sublinear growth of global skew with respect to the diameter.

Because of additional external noise and interference, one shall not rely on broadcasting algorithms alone. In practice, minimizing the broadcast time is only one side of the medal, reaching most nodes is important as well. Indeed, if a bridging node misses out a pulse, all nodes on the wrong side of the bridge will do as well. Missing several pulses will deteriorate the drift compensation, in particular if external conditions such as the weather have a high volatility. For these reasons one adds a pulling mechanism to the broadcast

algorithm. If a node does not receive the time information during a pulse, it will actively ask its neighbors. The reliability of this approach is apparent from the results provided in Section 6, as it solved the problem for the examined line topology: Sending a positive acknowledgement on reception of a message corresponds to sending a request—i.e., a negative acknowledgement—when a message is overdue.

7.2 Local Skew

So far we restricted our attention to the global skew, leaving the local skew aside. This comes from the fact that the fast propagation of clock estimates attained by the PulseSync algorithm in some sense makes the global skew "local". While conventional synchronization algorithms rely on techniques where information propagates at merely one hop per beacon interval, PulseSync floods it through the whole network within a single pulse. Thus, global skew cannot be accumulated for $\Omega(DB)$ time before recognized by distant nodes. Nonetheless, it grows like $\sqrt{D}$ due to the increasing (total) uncertainty in delays due to the jitter, whereas apparently neighbors are able to synchronize both their clocks and clock rates more precisely. The tree-like communication structure of PulseSync does not exploit this knowledge and therefore the full global skew may be observed between neighboring nodes which are at distance D (or even $2D$) in the tree.

Opposed to that, GTSP [29] exploits all local information by construction, as it strives to *locally* optimize skews in each step. However, this comes at the prize of slow dissemination of information, therefore incurring a large global skew. In theory, this discrepancy between optimizing locally and globally can be overcome easily. If we do not care about message complexity, but just frequency of pulses, *every* node could act as root of its own flooding tree. Clock values could then be computed as, say, (weighted) average over estimates of all nodes, which then would both locally and globally agree very well. This approach would require the unacceptable number of n^2 messages per pulse. One could also do with $O(n)$ messages; the root could collect information on the clock skew on each *edge* in the graph by means of a flooding-echo protocol. This comes at message cost $2n$ since a single message can be received by all neighbors simultaneously. The root then computes optimal skew and rate corrections in a centralized manner, where "optimal" can be any desired quality measure: Global skew, local skew, sum of squared errors on each edge (like in [28]), combinations of those, etc. Finally the obtained corrections are distributed by means of a second flooding. Though (up to constants) optimal with regard to skews and number of messages, obviously the size of the messages would become prohibitively large when relying on such a scheme.

However, for specific topologies similar approaches can be feasible. On a ring topology, e.g., one could simply use two root nodes r and s triggering pulses, located at opposite sides of the ring. One of the nodes, say s, then synchronizes itself with respect to r, while any other node v tries to synchronize to a convex combination of the received estimates $\hat{L}_r$ and $\hat{L}_s$, weighting them as $d(v,r)/d(r,s)$ and $d(v,s)/d(r,s)$ ($d(\cdot,\cdot)$ denoting distance), respectively. Adjustments to logical clock offsets are amortized over B time to avoid inducing local skews without need due to sudden corrections. Since the (global) skew between r and s will be "distributed" over paths of length $D/2$, the local skew would be improved to $O(\mathcal{J}\sqrt{(\ln n)/k})$, which again is asymptotically optimum. Admittedly, for medium diameters the benefit would be small though, as can be seen from a comparison of local and global skews in Figures 5 and 7. The line topology does not suffer from increased local skews due to a large stretch, yet the local skews are merely by a factor of roughly 2 better. Considering the factor two overhead in message and memory complexity compared to the plain PulseSync algorithm and its very small global skew, the gain seems limited for reasonable diameters.

8 Conclusions

In this work, we derived the novel clock synchronization algorithm PulseSync from a new theoretical analysis of the problem. It turned out that hitherto proposed protocols suffer from a global skew at least linear in the diameter. Even worse, the state-of-the-art Flooding Time Synchronization Protocol incurs skew exponential in the diameter. In contrast, we proved that PulseSync guarantees with high probability a maximum clock difference proportional to $\sqrt{D}$ at the same communication cost as FTSP. Furthermore, we provided a lower bound showing its asymptotic optimality, assuming that no outdated information is to be used.

We substantiated our theoretical findings by means of extensive simulations and practical experiments on a Mica2 testbed. Both simulations and experiments are in sound accordance with expectations. In particular, on a line topology of 20 sensor nodes PulseSync features maximum and average global skew of less than 40 and 5 microseconds, respectively, which is approximately a factor 5 better than the same values for FTSP. This serious scalability issue of FTSP is confirmed by the simulations, which clearly show the exponentially growing error of FTSP opposed to the root-like behavior of PulseSync. With 50 nodes, FTSP will experience *average* clock skews in the order of seconds, whereas it is reasonable to expect at most 80 microseconds skew between *any* two nodes when utilizing PulseSync.

Apart from these basic characteristics, PulseSync exhibits further desirable properties. It is able to adapt very quickly to any kind of dynamics, as only clock values from the last few pulses are incorporated in the calculation of logical clock values. Current algorithms will need a factor D more time to react to changes. Moreover, the accuracy provided does not depend on the clock drift nor on the frequency of communication, as long as clock rates do not change fast compared to the time between pulses. This means that—depending on the stability of environmental conditions—time intervals between pulses may be extended notably.

9 Acknowledgments

This work was partially supported by the Swiss National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS). We would like to thank Thomas Locher and Subhash Suri for their valuable input which helped us to improve the paper.

10 References

[1] TinyOS. http://www.tinyos.net/.

[2] M. Allen, L. Girod, R. Newton, S. Madden, D. T. Blumstein, and D. Estrin. VoxNet: An Interactive, Rapidly-Deployable Acoustic Monitoring Platform. In *Proc. 7th International Conference on Information Processing in Sensor Networks (IPSN)*, 2008.

[3] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the Time-Complexity of Broadcast in Multi-hop Radio Networks: An Exponential Gap Between Determinism and Randomization. *J. Comput. Syst. Sci.*, 45(1):104–126, 1992.

[4] S. Biaz and J. L. Welch. Closed Form Bounds for Clock Synchronization Under Simple Uncertainty Assumptions. *Inf. Process. Lett.*, 80(3):151–157, 2001.

[5] I. Chlamtac and S. Kutten. On Broadcasting in Radio Networks - Problem Analysis and Protocol Design. *IEEE Transactions on Communications*, 33:1240–1246, 1985.

[6] I. Chlamtac and O. Weinstein. The Wave Expansion Approach to Broadcasting in Multihop Radio Networks. In *Proc. 6th annual IEEE Conference on Computer Communications (INFOCOM)*, 1987.

[7] A. Czumaj and W. Rytter. Broadcasting Algorithms in Radio Networks with Unknown Topology. In *Proc. 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2003.

[8] J. Elson, L. Girod, and D. Estrin. Fine-Grained Network Time Synchronization using Reference Broadcasts. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[9] R. Fan and N. Lynch. Gradient Clock Synchronization. In *Proc. 23rd annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2004.

[10] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-Sync Protocol for Sensor Networks. In *Proc. 1st International Conference on Embedded Networked Sensor Systems (SenSys)*, 2003.

[11] J. Hill and D. Culler. Mica: a Wireless Platform for Deeply Embedded Networks. *Micro, IEEE*, 22(6):12–24, 2002.

[12] W. Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[13] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health Monitoring of Civil Infrastructures using Wireless Sensor Networks. In *Proc. 6th International Conference on Information Processing in Sensor Networks (IPSN)*, 2007.

[14] H. Kopetz and W. Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Trans. Comput.*, 36(8), 1987.

[15] F. Kuhn and R. Oshman. Gradient Clock Synchronization using Reference Broadcasts. *CoRR*, abs/0905.3454, 2009.

[16] B. Kusy. *Spatiotemporal Coordination in Wireless Sensor Networks*. PhD thesis, Vanderbilt University, 2007.

[17] C. Lenzen, T. Locher, and R. Wattenhofer. Tight Bounds for Clock Synchronization. In *Proc. 28rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2009.

[18] J. Lundelius and N. A. Lynch. An Upper and Lower Bound for Clock Synchronization. *Information and Control*, 62(2/3):190–204, 1984.

[19] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The Flooding Time Synchronization Protocol. In *Proc. 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[20] L. Meier and L. Thiele. Brief announcement: Gradient Clock Synchronization in Sensor Networks. In *Proc. 24th annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2005.

[21] D. Mills. Internet Time Synchronization: the Network Time Protocol. *IEEE Transactions on Communications*, 39(10):1482–1493, Oct 1991.

[22] T. Moscibroda, P. von Rickenbach, and R. Wattenhofer. Analyzing the Energy-Latency Trade-Off During the Deployment of Sensor Networks. In *Proc. 25th Conference on Computer Communications (INFOCOM)*, 2006.

[23] R. Ostrovsky and B. Patt-Shamir. Optimal and Efficient Clock Synchronization Under drifting Clocks. In *Proc. 18th annual ACM Symposium on Principles of Distributed Computing (PODC)*, 1999.

[24] K. Römer. Time Synchronization in Ad Hoc Networks. In *Proc. 2nd ACM International Symposium on Mobile ad hoc Networking & Computing (MobiHoc)*, 2001.

[25] J. Sallai, B. Kusy, A. Ledeczi, and P. Dutta. On the Scalability of Routing Integrated Time Synchronization. *3rd European Workshop on Wireless Sensor Networks (EWSN)*, 2006.

[26] J. Schneider and R. Wattenhofer. A log-star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *Proc. 27th ACM Symposium on Principles of Distributed Computing (PODC)*, 2008.

[27] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor Network-based Countersniper System. In *Proc. 2nd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[28] R. Solis, V. Borkar, and P. R. Kumar. A New Distributed Time Synchronization Protocol for Multihop Wireless Networks. In *Proc. 45th IEEE Conference on Decision and Control (CDC)*, 2006.

[29] P. Sommer and R. Wattenhofer. Gradient Clock Synchronization in Wireless Sensor Networks. In *Proc. 8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, 2009.

[30] T. K. Srikanth and S. Toueg. Optimal Clock Synchronization. *J. ACM*, 34(3), 1987.

[31] A. A. Syed and J. Heidemann. Time synchronization for high latency acoustic networks. In *Proc. 25th Conference on Computer Communications (InfoCom)*, 2006.

[32] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and Yield in a Volcano Monitoring Sensor Network. In *Proc. 7th Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[33] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal. Firefly-Inspired Sensor Network Synchronicity with Realistic Radio Effects. In *Proc. 3rd International Conference on Embedded Networked Sensor Systems (SenSys)*, 2005.