

Tiny-Sync: Tight Time Synchronization for Wireless Sensor Networks

SUYOUNG YOON, CHANCHAI VEERARITTIIPHAN, and MIHAIL L. SICHITIU

Dept. of Electrical and Computer Engineering,
North Carolina State University,
Raleigh, NC 27695

Time synchronization is a fundamental middleware service for any distributed system. Wireless sensor networks make extensive use of synchronized time in many contexts (e.g., data fusion, TDMA schedules, synchronized sleep periods, etc.). We propose a time synchronization method relevant for wireless sensor networks. The solution features minimal complexity in network bandwidth, storage and processing, and can achieve good accuracy. Especially relevant for sensor networks, it also provides tight, deterministic bounds on offset and clock drift. A method for synchronizing the entire network is presented. The performance of the algorithm is analyzed theoretically and is validated on a realistic testbed. The results show that the proposed algorithm outperforms existing algorithms in terms of precision and resource requirements.

Categories and Subject Descriptors: C.2.4 [**COMPUTER-COMMUNICATION NETWORKS**]: Distributed Systems; C.2.2 [**COMPUTER-COMMUNICATION NETWORKS**]: Network Protocols

General Terms: Algorithm, Design, Experimentation, Performance

Additional Key Words and Phrases: Sensor networks, Time synchronization

1. INTRODUCTION

Recent technological advances in low power radios, sensors and microcontrollers enable a new monitoring paradigm for large geographical areas. The vision involves a large number of inexpensive nodes equipped with one or more sensors, a small microcontroller and transceivers capable of short-range communications. Wireless sensor networks (WSNs) [Akyildiz et al. 2002a; 2002b; Asada et al. 1998] have the potential to truly revolutionize the way we monitor and control our environment.

Many (if not most) WSN applications either benefit from, or require, time synchronization. Sensed data is typically forwarded over multiple hops to one or more base stations and encounters delays ranging from a few tens of milliseconds to several minutes [Xu et al. 2004]. Many energy-efficient algorithms trade power savings for increased delays. Therefore, without time synchronization, the time that sensed

Author's address: Suyoung Yoon, Chanchai Veerarittiphan and Mihail L. Sichitiu, Dept. of Electrical and Computer Eng. North Carolina State University Raleigh, NC 27695-7911.

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0000-0000/20YY/0000-0001 \$5.00

data reached the base station is often a poor approximation of the time the data was sensed. If the entire sensor network is time-synchronized (with respect to the base station), meaningful time-stamps can accompany sensed data. Target tracking, one of the most popular sensor network applications, requires tight time synchronization for beam-forming. Many WSN protocols (e.g., power-saving sleep schedules [Lu et al. 2005], TDMA schedules for maximizing the networks' capacity, security mechanisms for preventing replay attacks, etc.) also rely on time synchronization.

In this paper, we present two closely-related algorithms suitable for *pair-wise* time synchronization. In those algorithms, one node determines (tight) bounds of the relative offset and drift of its clock with respect to the one of the other node. We also present a scheme building on these algorithms, which is capable of synchronizing the clocks of an entire sensor field. To avoid confusion between the two algorithms, we will name them *mini-sync* and *tiny-sync* as they use limited and very limited resources, respectively. The two algorithms feature:

- Drift awareness*: The algorithms not only take the drift of the clock into account, but also find tight bounds on the drift.
- Deterministic bounds on the precision*: Most other algorithms provide best estimates for the offset and drift of the clocks, and possibly probabilistic bounds on these estimates. Our approach delivers tight, *deterministic* bounds on these estimates, such that absolute information can be deduced about ordering and simultaneous events.
- Precision*: Given small uncertainty bounds on the delays exchanged messages undergo, the precision of the synchronization can be arbitrarily good.
- Low computation and storage complexity*: Wireless sensor nodes typically feature low computational power microcontrollers with small amounts of RAM. Both algorithms have low computational and storage complexity.
- Low sensitivity to communication errors*: Wireless communications are notoriously error prone, and thus one cannot rely on correct receipt of all messages. The presented approach works correctly even if a large percentage of the messages is lost.
- Hardware independence*: The two-way data collection exchange does not require hardware-dependent calibration, and allows the proposed algorithms to detect and discard outliers.

The main contribution of the paper is the development and performance evaluation of two *simple* algorithms that deliver accurate offset and drift information together with *tight* bounds on them. In short, the two algorithms select a bounding box on the relative drift and offset that, in most cases, outperforms linear regression by rejecting outliers while continuously monitoring and adapting to clock non-linearity.

The two algorithms presented in this paper are not limited to wireless sensor networks. They can synchronize the nodes on any communication network which allows bidirectional data transmission. However, the algorithms provide very good precision (microsecond if crafted carefully) and bounds on the precision while using very limited resources, thus being especially well suited for wireless sensor networks.

The rest of the paper is organized as follows. Section 2 presents a brief overview of the related work. In Section 3, two time synchronization algorithms are presented and their theoretical performance is analyzed. Section 4 presents the experimental results on the performance of the two proposed algorithms. Section 6 concludes the paper.

2. RELATED WORK

Time synchronization is a key service for many applications and operating systems in distributed computing environments. Many protocols have been proposed and used for time synchronization in wired and wireless networks. Mill’s Network Time Protocol (NTP) [Mills 1991; 1994] has been widely used in the Internet for decades. Nodes could also be equipped with a global positioning system (GPS) [Hofmann-Wellenhof et al. 1997] to synchronize them. However, traditional synchronization schemes and GPS-equipped systems are not suitable for use in WSNs due to the specific requirements of those networks:

- Precision*: Depending on the considered application, WSNs may require far better precision than traditional networks. For example, a precision of a few milliseconds is considered satisfactory for NTP, while in a WSN beam-forming application, microsecond precision can significantly improve the performance of the application [Simon et al. 2004]. Furthermore, when coordinating synchronized time schedules, the higher the precision of the synchronization algorithm, the smaller the guard times have to be (and, hence, the higher the efficiency of the scheduling approach).
- Cost*: Cost is of primary concern in WSNs as, typically, nodes have limited batteries, computational and storage resources. Most of the protocols designed for wired environments exchange many messages for statistical processing. Furthermore, the protocols also need to store the messages to process them.

Recently, a significant amount of research on time synchronization for wireless sensor networks has been published [Sivrikaya and Yener 2004; Sundararaman et al. 2005]. An interesting approach called *post facto synchronization* was proposed by Elson and Estrin [Elson and Estrin 2001]. In this approach, each node’s clock is normally unsynchronized with the rest of the network; a beacon node periodically broadcasts beacon messages to the sensor nodes in its wireless range. When an event is detected, each node records the time of the event (time-stamp with its own local clock). After the event (hence, the name), upon receiving the reference beacon message, nodes use it as time reference and adjust their event timestamps with respect to that reference.

The reference broadcast synchronization (RBS) protocol [Elson et al. 2002], uses a data collection mechanism similar to the one in the post facto synchronization: one node acts as a beacon by broadcasting a reference packet. All receivers record the packet arrival time. The receiver nodes then exchange their recorded timestamps and estimate their relative phase offsets. RBS also estimates the clock skew by using a least-squares linear regression. The interesting feature of RBS is that it records the timestamp only at the receivers. Thus, all timing uncertainties (including MAC medium access time) on the transmitter’s side are eliminated. This characteristic

makes it especially suitable for hardware that does not provide low-level access to the MAC layer (e.g., 802.11). Although RBS synchronization only involves one hop neighbors, the mechanism can be extended to synchronize a multi-hop network [Karp et al. 2003]. In such a system, timestamps in messages can be reconciled as they are being forwarded by the intermediate nodes (according to their next-hop destination and its time difference to the current node) [Girod 2005]. The main disadvantage of RBS is that it does not synchronize the sender with the receiver directly and that, when the programmers have low-level access at the MAC layer, simpler methods (e.g., TPSN) can achieve a similar precision to RBS.

Römer [Römer 2001] presented a synchronization protocol for ad hoc networks. The authors assume clocks with known upper-bounds on the clock drift. The basic idea of the algorithm is to compute timestamps using unsynchronized local clocks. When a local time-stamp is transferred between two nodes, the timestamp is transformed to the local time of the receiving node with guaranteed bounds based on the assumed maximum clock drift. These protocols focus on temporal relationships between the events such as "event X happened before event Y" and "event X and Y happened within a certain time interval." [Meier et al. 2004] enhanced Römer's algorithm and achieved higher accuracy while reducing computation.

When implementing time synchronization protocols, significant challenge is minimizing timestamping uncertainties. RBS reduces these uncertainties by timestamping only at the receivers. The time synchronization protocol for sensor network (TPSN) [Ganerwal et al. 2003] reduces the uncertainties by using timestamps at the medium access control (MAC) layer. This eliminates the (large) uncertainties introduced by the MAC layer, e.g., retransmissions, backoffs, medium access, etc. TPSN takes advantage of the availability of the MAC layer code in TinyOS [Hill et al. 2000]. For a single beacon, TPSN offers a two-fold increase in precision in comparison to RBS (although, asymptotically, as more beacons are sent, they achieve the same precision).

The Flooding Time Synchronization Protocol (FTSP) [Maróti et al. 2004] was designed for a sniper localization application requiring very high precision [Simon et al. 2004]. FTSP achieves the required accuracy by utilizing a customized MAC layer time-stamping and by using calibration to eliminate unknown delays. FTSP is robust to network failures, as it uses flooding both for pair-wise and for global synchronization. Linear regression from multiple timestamps is used to estimate the clock drift and offset. The main drawback of FTSP is that it requires calibration on the hardware actually used in the deployment (and thus, it is not a purely software solution independent of the hardware). FTSP also *requires* intimate access to the MAC layer for multiple timestamps. However, if well calibrated, the FTSP's precision is impressive (less than $2\mu s$).

Lightweight Time Synchronization (LTS) [Greunen and Rabaey 2003] was proposed for applications where the required time accuracy is relatively low. The pair-wise synchronization on LTS is similar to TPSN except for the treatment of the uncertainties (LTS adopts a statistical model for handling the errors). The simulation results show that the accuracy of LTS is about 0.5 seconds.

Li and Rus [Li and Rus 2004] presented a high-level framework for global synchronization. The authors proposed three methods for global synchronization in

WSNs. The first two methods, all-node-based and cluster-based synchronization, use global information and are, hence, not suitable for large WSNs. In the third method (diffusion), each node sets its clock to the average clock time of its neighbors. The authors showed that the diffusion method converges to a global average value. A drawback of this approach is the potentially large number of messages exchanged between neighbor nodes, especially in dense networks.

Dai and Han [Dai and Han 2004] proposed two time synchronization protocols, Hierarchy Referencing Time Synchronization (HRTS) and Individual-based Time Request (ITR). HRTS uses an idea similar to RBS except for the synchronization of the receivers after the beacon synchronization message was sent: instead of the receivers exchanging messages among themselves, a designated node sends its time to the beacon node that, in turn, broadcasts this message over the entire network, thus, significantly reducing the number of message exchanges. Similar to RBS, the beacon node has to be able to broadcast to the entire sensor network. The ITR protocol is based on NTP. Multichannel support is integrated both in ITR as well as HRTS to reduce the delay variations.

Delay Measurement Time Synchronization Protocol (DMTS) [Ping 2003] reduces the number of message exchanges in RBS by accurately estimating the delay of the path from the sender to the receiver. DMTS is similar to RBS on the sender side and TPSN or FTSP on the receiver side.

Adaptive Clock Synchronization [PalChaudhuri et al. 2004] is a probabilistic method for clock synchronization that uses the higher precision of receiver-to-receiver synchronization. The protocol extended the deterministic RBS protocol to provide a probabilistic bound on the accuracy of the clock synchronization. The bound allows a tradeoff between the accuracy and the resource requirement.

Su [Su and Akyildiz 2005] proposed Time-Diffusion Synchronization Protocol (TDP) for network-wide time synchronization. TDP maintains global time synchronization within an adjustable bound (based on the application requirements). TDP achieves global synchronization by multi-hop flooding: the base station initiates the protocol by sending a special timing message to the entire network. Some of the nodes, upon receiving the message, become masters by using a leader election procedure (that uses a False Ticker Isolation Algorithm to discard outliers and a Load Distribution Algorithm to balance the energy consumption of the network). The master nodes start the time diffusion procedure involving electing diffused leaders (similar to the master election algorithm), multi-hop flooding and iterative weighted averaging of timing from different master nodes. TDP handles node mobility and failures by using a Peer Evaluation Procedure. The method achieves a precision of 0.1s.

3. PROPOSED ALGORITHMS

In this section, we present the two proposed algorithms and their expected performance.

3.1 Data Collection

For the proposed algorithms, we will use a classical data collection algorithm [Ganerival et al. 2003; Cristian 1989; Mills 1991; Lemmon et al. 2000; Römer 2001]; however, we will process the data differently.

Consider a wireless node i with its hardware clock $t_i(t)$, where t is the Coordinated Universal Time (UTC). In general, the hardware clock of node i is a monotonically non-decreasing function of t . In practice, a quartz oscillator is often used to generate the real-time clock. The oscillator's frequency depends on the ambient conditions, but for relatively extended periods of time (minutes - hours), the hardware clock can be approximated with good accuracy by an oscillator with fixed frequency:

$$t_i(t) = a_i t + b_i, \quad (1)$$

where a_i and b_i are the drift and the offset of node i 's clock. In general, a_i and b_i will be different for each node and approximately constant for an extended period of time.

Consider two wireless nodes 1 and 2 with their hardware clocks $t_1(t)$ and $t_2(t)$, respectively. From (1), it follows that t_1 and t_2 are linearly related:

$$t_1(t) = a_{12} t_2(t) + b_{12}. \quad (2)$$

The parameters a_{12} and b_{12} represent the *relative drift* and the *relative offset* between the two clocks, respectively. If the two clocks are perfectly synchronized, the relative drift is equal to one, and the relative offset is equal to zero.

Assume that node 1 would like to be able to determine the relationship between t_1 and t_2 . Node 1 sends a probe message to node 2. The probe message is time-stamped just before it is sent with t_o . Upon receipt, node 2 time-stamps the probe t_b and returns it immediately (we will shortly relax this constraint) to node 1 which timestamps it upon receipt t_r . Figure 1 depicts such an exchange.

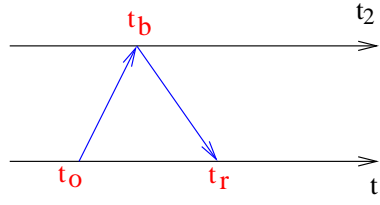


Fig. 1. A probe message from node 1 is immediately returned by node 2 and time-stamped at each send/receive point resulting in the data-point (t_o, t_b, t_r) .

The three time-stamps (t_o, t_b, t_r) form a data-point which effectively limits the possible values of parameters a_{12} and b_{12} in (2). Indeed, since t_o happened *before* t_b , and t_b happened *before* t_r , the following inequalities should hold:

$$t_o < a_{12} t_b + b_{12}, \quad \text{and} \quad (3)$$

$$t_r > a_{12} t_b + b_{12}. \quad (4)$$

The data collection procedure described above is repeated several times; and each probe that returns, provides a new data-point and, thus, new constraints on the admissible values of a_{12} and b_{12} .

The linear dependence between t_1 and t_2 and the constraints imposed by the data-points can be represented graphically as shown in Fig. 2. Each data-point can

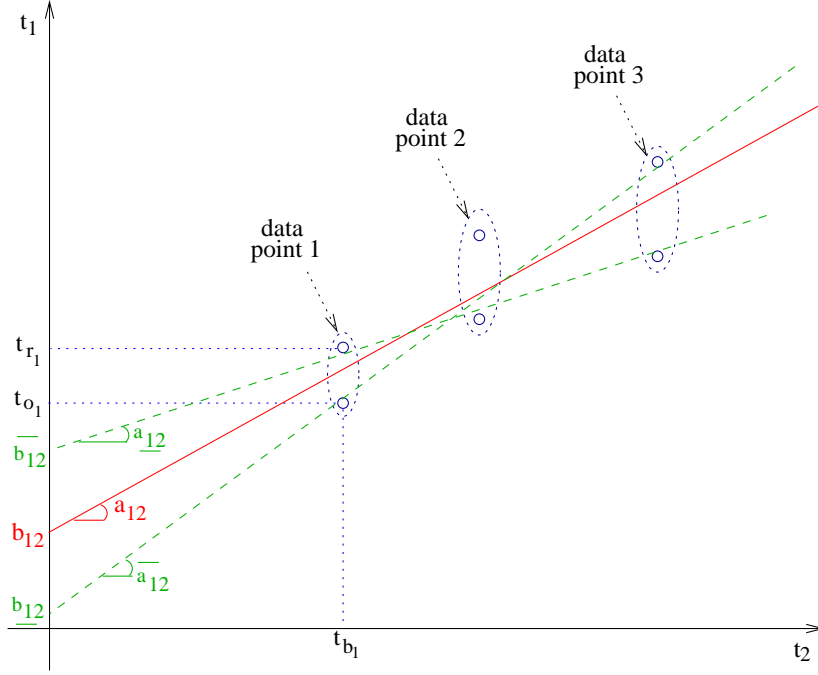


Fig. 2. The linear dependence (2) and the constraints imposed on a_{12} and b_{12} by three data-points.

be represented by two constraints in the system of coordinates given by the local clocks of the two nodes t_2 and t_1 . The double subscript of the timestamps t_{o_i} , t_{b_i} and t_{r_i} denotes the timestamps corresponding to data-point i . Inequality (3) imposes a constraint on the line representing the relationship (2): the line has to be over the point of coordinates (t_b, t_o) . Similarly, corresponding to inequality (4), the line has to be under the point of coordinates (t_b, t_r) (the second constraint). Satisfying both constraints, requires the line to be positioned *between* the two constraints determined by each data-point. The exact values of a_{12} and b_{12} cannot be accurately determined using this approach (or any other approach) as long as the message delays are unknown. But, a_{12} and b_{12} can be bounded by:

$$\underline{a_{12}} \leq a_{12} \leq \overline{a_{12}}, \quad \text{and} \quad (5)$$

$$\underline{b_{12}} \leq b_{12} \leq \overline{b_{12}}, \quad (6)$$

where $\overline{a_{12}}$ ($\underline{a_{12}}$) is the maximum (minimum) of the slopes of lines that satisfy the constraints, and $\overline{b_{12}}$ ($\underline{b_{12}}$) is the value on the y-axis at the intersection with the line corresponding to $\overline{a_{12}}$ ($\underline{a_{12}}$).

Not all combinations of a_{12} and b_{12} satisfying (5) and (6) are valid, but all valid combinations satisfy (5) and (6). The real values of a_{12} and b_{12} can be estimated as the midpoint of the range of possible values $\widehat{a_{12}}$ and $\widehat{b_{12}}$:

$$a_{12} \in \left[\widehat{a_{12}} - \frac{\Delta a_{12}}{2}; \widehat{a_{12}} + \frac{\Delta a_{12}}{2} \right], \quad \text{and} \quad (7)$$

$$b_{12} \in \left[\widehat{b}_{12} - \frac{\Delta b_{12}}{2}; \widehat{b}_{12} + \frac{\Delta b_{12}}{2} \right], \quad (8)$$

where

$$\widehat{a}_{12} = \frac{\overline{a}_{12} + \underline{a}_{12}}{2}, \quad (9)$$

$$\Delta a_{12} = \overline{a}_{12} - \underline{a}_{12}, \quad (10)$$

$$\widehat{b}_{12} = \frac{\overline{b}_{12} + \underline{b}_{12}}{2}, \quad \text{and} \quad (11)$$

$$\Delta b_{12} = \overline{b}_{12} - \underline{b}_{12}. \quad (12)$$

The goal of the algorithms is to determine \overline{a}_{12} , \underline{a}_{12} , \overline{b}_{12} and \underline{b}_{12} as tight as possible (such that it minimizes Δa_{12} and Δb_{12}). Once a_{12} and b_{12} are estimated, node 1 can always correct the reading of the local clock (using (2)) to have it match the readings of the clock at node 2.

To decrease the overhead of this data-gathering algorithm, the probes can be piggy-backed on data messages. Since most MAC protocols in wireless networks employ an acknowledgment (ACK) scheme, the probes can be piggy-backed on the data and the responses on the ACKs. Elaborate schemes with optional headers can be devised to reduce the length of the header when probes do not need to be sent. This way, synchronization can be achieved almost “for free” (i.e., with very little overhead in terms of communication bandwidth).

3.1.1 Relaxing the Immediate Reply Assumption. In Fig. 1, we assumed that node 2 replies immediately to node 1 when it receives a probe. The correctness of the presented approach is not affected in any way even if node 2 does not respond *immediately*. Node 2 can delay the reply as long as it wants; the relations (3) and (4), and, thus the rest of the analysis will still hold.

However, as the delay between t_o and t_r increases, the precision of the estimates will decrease. In practice, node 2 may have to delay the reply due to any number of reasons (e.g., it has something more important to send, it has to postpone its transmission due to medium access contention, etc.).

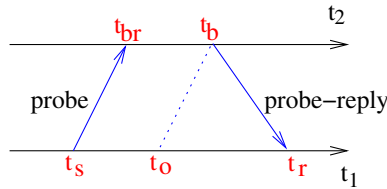


Fig. 3. A probe message from node 1 may be returned by node 2 after being time-stamped at both the send and receive points.

To counteract the possible loss in precision, node 2 can time-stamp the probe message upon receipt (t_{br}) as well as upon reply (t_b) (as depicted in Fig. 3). In this case, node 1 can adjust t_o as follows:

$$t_o = t_s + \widehat{a}_{12}(t_b - t_{br}), \quad (13)$$

where t_o is the latest time at node 1 that is known to have occurred before t_b . The same inequalities (3) and (4) hold for this case as well.

3.1.2 Increasing Precision by Considering Minimum Delay. If no information about delays encountered by the probe messages is available, nothing else can be done to increase the precision. However, if the minimum delay a probe encounters between the nodes is known, the data-points can be adjusted for an increase in the precision of the results.

To determine the minimum delay, one can take into account the minimum length of such a probe and the time it takes to transmit such a probe (at the transmission rate of the sensor node), and, eventually, other operations that have to be completed before the probe is sent or upon receiving such a probe (e.g., encryption/decryption, CRC calculation, etc.).

Assume that we are able to determine the minimum delay δ_{12} that the probe encounters between t_o (t_s) at node 1 and t_b (t_{br}) at node 2. Also, denote with δ_{21} , the minimum delay between the moment t_b at node 2 and the moment t_r at node 1. Then, $(t_o + \delta_{12}, t_b, t_r - \delta_{21})$ should be used as a data-point as this will offer increased accuracy over the data-point (t_o, t_b, t_r) .

If, for two probes, both minimums δ_{12} and δ_{21} are reached, the method presented in this paper can achieve *perfect* synchronization (i.e., there will be no uncertainties for relative offset and relative drift: $\Delta a_{12} = \Delta b_{12} = 0$).

3.2 Tiny-sync and Mini-sync - Processing the Data

After acquiring a few (at least two) data-points, the offset and the drift can be estimated using inequalities (3) and (4). An existing solution for finding the optimal bounds on the drift and offset involves solving two linear programming problems with twice as many inequalities as data-points [Lemmon et al. 2000]. By optimal bounds (and the corresponding optimal solution) we mean the bounds that minimize the difference between the bounds.

The disadvantage of this approach is that as more and more data samples are collected, the computational and storage requirements increase (potentially unbounded). Also, one should not limit the number of collected samples to a fixed window as the best drift estimates are obtained when a large number of samples are available. The approach in [Lemmon et al. 2000] is clearly not suitable for systems with limited memory and computing resources such as wireless sensor nodes. In this paper, we will pursue another avenue.

The two proposed algorithms spring from the observation that not all data-points are useful. In Fig. 2, the bounds on the estimates $[a_{12}, \overline{a_{12}}]$ and $[b_{12}, \overline{b_{12}}]$ are constrained only by data-points 1 and 3. Therefore, we do not need data-point 2, and we can discard it, as data-point 3 produces better estimates than data-point 2.

It seems that only four constraints (the ones which define the best bounds on the estimates) have to be stored at any time. Upon the arrival of a new data-point, the two new constraints are compared with the existing four constraints and two of the six are discarded (i.e., the four constraints which result in the best estimates are kept). The comparison operation to decide which four constraints to be kept is very simple, computationally (only 8 additions, 4 divisions and 4 comparisons). At any one time, only the information for the best four constraints needs to be

stored. We will name the algorithm described in this paragraph “tiny-sync.” The four constraints that are stored at any one time instant may belong to two, three or four different data-points. With the four constraints, the bounds of the clock offset and drift are easily computed from the two lines that can be constructed with the two lower bound and two upper bound constraints: the first line is determined by the first lower bound constraint and the second upper bound constraint and the other line is determined by the first upper bound constraint and second lower bound constraint. The values of \bar{a}_{12} , \underline{b}_{12} , \underline{a}_{12} and \bar{b}_{12} are the slope and offset of those two lines respectively.

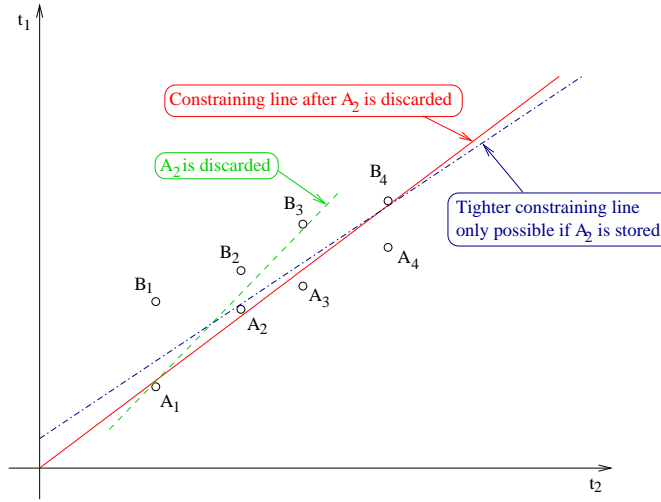


Fig. 4. In this situation, after receiving data-point $A_3 - B_3$, tiny-sync will discard constraints A_2 and B_2 . However, after receiving $A_4 - B_4$, it turns out that the most constraining constraint would have been $A_2 - B_4$.

Unfortunately, while tiny-sync is very efficient, it does not always produce the optimal solution. Consider the situation depicted in Fig. 4. For clarity, we labeled each constraint individually, that is, for data-point i , A_i represents constraint (t_{b_i}, t_{o_i}) , and B_i represents constraint (t_{b_i}, t_{r_i}) . After the first two data-points (A_1-B_1) and (A_2-B_2) are received, the first estimates for the drift and offset may be computed. After the third data-point (A_3-B_3) is received, the bounds on the estimates improve (i.e., Δa_{12} and Δb_{12} are smaller); so, the constraints A_1, B_1, A_3 and B_3 are stored, while A_2 and B_2 are discarded. The next data point (A_4-B_4) could have used constraint A_2 to construct a better estimate. Unfortunately, A_2 was already discarded at this point; and thus, a less than ideal estimate for \underline{b}_{12} will now be imposed by A_1 and A_4 . Thus, while producing correct results, tiny-sync might miss the optimum result. We will compare the performance of tiny-sync with the optimal solution in Section 4.

In Fig. 4, the constraint A_2 was discarded by tiny-sync because it was not immediately useful, but rather only potentially useful in the future. By a potentially useful constraint we mean a constraint that, with the current data points, is not one

of the constraints that is determining the values of \underline{a}_{12} , \overline{a}_{12} , \underline{b}_{12} and \overline{b}_{12} , but which may do so given future data points. This does not mean that all the constraints are potentially useful. In fact, only the constraints A_j (e.g., A_2) that satisfy the condition

$$m(A_i, A_j) > m(A_j, A_k) \quad (14)$$

for some integers $1 \leq i < j < k$ are potentially useful in the future (by $m(X, Y)$, we denote the slope of the line going through the points X and Y).

Theorem 1 *Any constraint A_j which satisfies*

$$m(A_i, A_j) \leq m(A_j, A_k) \quad (15)$$

for at least one set of integers $1 \leq i < j < k$ can be safely discarded as it will never constrain the bounds \underline{a}_{12} , \overline{a}_{12} , \underline{b}_{12} and \overline{b}_{12} more than any existing constraints.

The proof is presented in the Appendix. Similar conditions for discarding upper-bound constraints (B_i) exist.

The resulting algorithm (called “mini-sync”), upon the receipt of a new-data point, will check if the new constraints can eliminate any of the old constraints. Potentially, many old constraints can be eliminated with one new data-point. We use all the remaining constraints to obtain the (still optimal) solution using the same procedure as in [Lemmon et al. 2000]. Since we only eliminate constraints (inequalities) that are irrelevant, we still obtain the optimal solution with only a few points (solving the set of all inequalities is shown to result in the optimal solution [Lemmon et al. 2000]).

Storing only four points, as in tiny-sync, does not produce the optimal solution. How many points have to be stored for the optimal solution? Theoretically, a potentially large number. If the delay between nodes 1 and 2 is monotonically increasing, (14) can hold for all of the constraints A_j . In practice, the delays do not continuously increase monotonically; therefore, only several constraints need to be stored to obtain the optimal result.

3.3 Analysis of the Proposed Algorithms

In this section, we will analyze the expected performance of tiny-sync as a function of the system parameters (round-trip time, probing period, etc.).

For the analysis, we make the simplifying assumption that the difference between t_o and t_r is constant and equal to RTT :

$$t_{r_i} - t_{o_i} = RTT \quad \forall i \geq 1. \quad (16)$$

In other words, we assume that the sum of all unknown delays between the moment the probe is sent and the moment the reply is received is always constant and equal to RTT .

We justify this simplifying assumption by considering the following:

- Tiny-sync only stores and uses in its computation the *best* two data-points collected so far (in terms of constraining the uncertain relative drift and offset). Data points with small round-trip times result in the tightest constraints, and, hence, most often, the two stored data-points have their round-trip times equal to the minimum round trip-time of the connection;

- The *variation* of the round-trip delays is typically very small (microseconds) when compared to the sampling intervals (seconds to tens of seconds); and
- As shown in Section 4.2, the theoretical analysis matches the experimental results exceedingly well.

With assumption (16), the data points that define the best approximation are always the first and last data-points collected. All of the other data-points can be safely discarded. Thus, with this assumption, tiny-sync’s performance is identical to the performance of mini-sync (the optimal algorithm). Denote with $(t_{o_1}, t_{b_1}, t_{r_1})$ and $(t_{o_2}, t_{b_2}, t_{r_2})$ the first and last collected data-points, respectively.

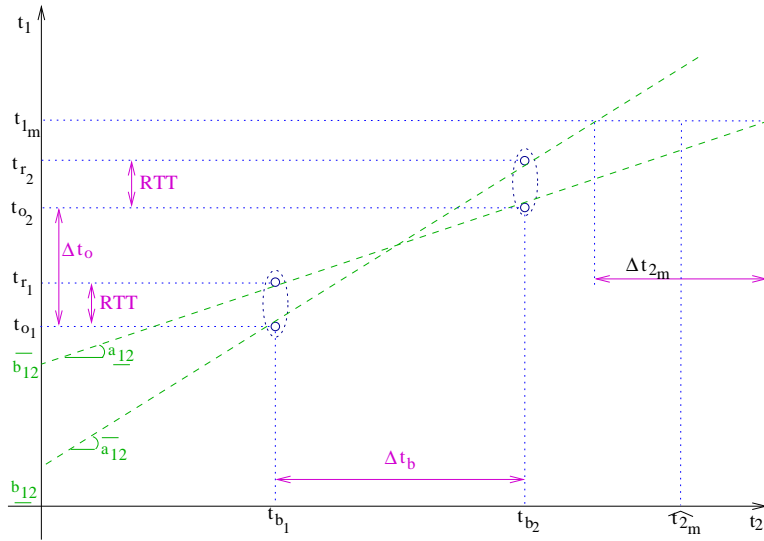


Fig. 5. Setup for the simplified analysis of the algorithms.

We use the following notation:

$$\Delta t_o = t_{o_2} - t_{o_1} \quad \text{and} \quad (17)$$

$$\Delta t_b = t_{b_2} - t_{b_1}. \quad (18)$$

Solving the linear equations for the unknowns \underline{a}_{12} , \overline{a}_{12} , \underline{b}_{12} and \overline{b}_{12} , we obtain:

$$\underline{a}_{12} = \frac{\Delta t_o - RTT}{\Delta t_b}, \quad (19)$$

$$\overline{a}_{12} = \frac{\Delta t_o + RTT}{\Delta t_b}, \quad (20)$$

$$\underline{b}_{12} = t_{o_1} - t_{b_1} \underline{a}_{12}, \quad \text{and} \quad (21)$$

$$\overline{b}_{12} = t_{o_1} + RTT - t_{b_1} \underline{a}_{12}. \quad (22)$$

Using (10) and (12), we obtain the following bounds on Δa_{12} and Δb_{12} :

$$\Delta a_{12} = \frac{2RTT}{\Delta t_b}, \quad \text{and} \quad (23)$$

$$\Delta b_{12} = RTT + t_{b_1} \Delta a_{12}. \quad (24)$$

Equations (23) and (24) capture the essential behavior of the presented algorithms. Equation (23) implies that the uncertainty bound on the drift decreases continuously as the distance between the first and the last collected data-points increases:

$$\lim_{\Delta t_b \rightarrow \infty} \Delta a_{12} = 0. \quad (25)$$

Figure 6 depicts the evolution of the uncertainty bound on the relative clock drifts Δa_{12} , which remains constant between data-points, and improves upon the receipt of a new data-point.

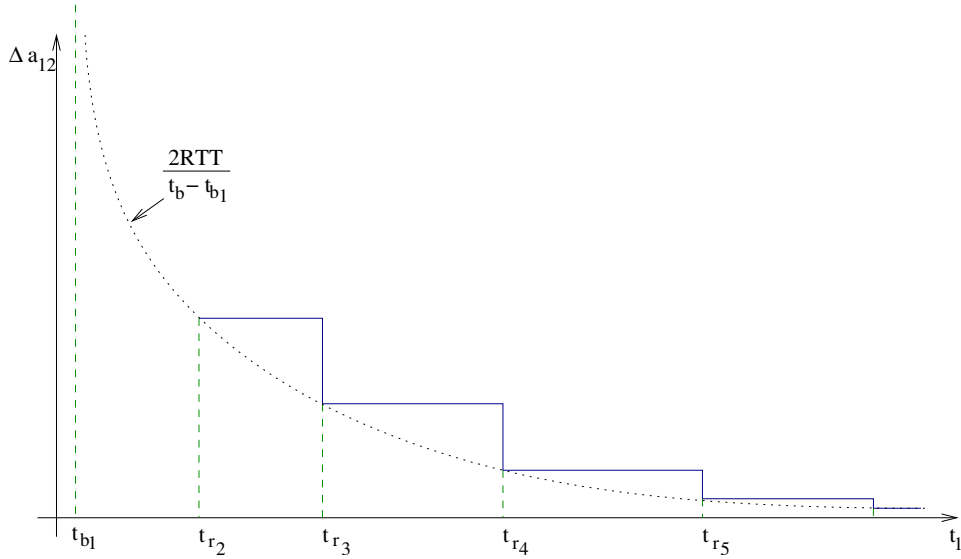


Fig. 6. Evolution of the uncertainty bound on the relative clock drifts Δa_{12} as new data-points are received.

Theoretically, the precision on the relative drift for the two clocks Δa_{12} can be made arbitrarily small. In practice, assumption (1) only holds for limited time T_0 , and, thus, the achievable precision is on the order of $\frac{2RTT}{T_0}$.

Equation (24) implies that the precision on the relative offset Δb_{12} is limited by the round-trip time RTT :

$$\lim_{\Delta t_b \rightarrow \infty} \Delta b_{12} = RTT. \quad (26)$$

In practice, if assumption (16) is eliminated, Δb_{12} is slightly higher than the minimum RTT . If MAC layer timestamping is possible and/or the minimum deterministic RTT delay is known (as described in Section 3.1.2), then the value of the RTT in (26) reduces accordingly (to account only for the non-deterministic component of the delay). In turn, this corresponds to an increase in the precision of the approach.

The second implication of (24) is that the performance of the algorithm depends on the choice of origin: if t_{b_1} is large, the precision of the offset will initially be poor (it will improve as $\Delta a_{12} \rightarrow 0$). To avoid the initial loss in precision, the origin can be shifted:

$$t_{b_1} = 0. \quad (27)$$

This shift can be easily achieved by subtracting t_{b_1} from every component of all data-points and keeping the rest of the algorithms unchanged.

The goal of the synchronization is to be able to estimate the clock of the remote machine t_2 given the local clock t_1 after the two clocks have been synchronized. In Fig. 5, node 1 estimates the value of node 2's clock t_{2_m} at the same real time that node 1's clock is reading t_{1_m} .

Using (2) and writing t_{2_m} as:

$$t_{2_m} = \widehat{t_{2_m}} \pm \frac{\Delta t_{2_m}}{2}, \quad (28)$$

we can find:

$$\Delta t_{2_m}(t_{1_m}) = t_{1_m} \frac{\Delta a_{12}}{\underline{a_{12}} \overline{a_{12}}} + \frac{a_{12} b_{12} - \overline{a_{12}} \overline{b_{12}}}{\underline{a_{12}} \underline{a_{12}}}. \quad (29)$$

Using (19)-(22), (29) becomes:

$$\Delta t_{2_m}(t_{1_m}) = \frac{\Delta a_{12}(t_{1_m} - t_{o_1})}{\underline{a_{12}} \underline{a_{12}}} + \frac{RTT}{\underline{a_{12}}}. \quad (30)$$

Fig. 7 depicts the qualitative evolution of the uncertainty bound Δt_{2_m} as more data-points are received, (i.e. a sketch of Δt_{2_m} as a function of t_{1_m} as in (30)).

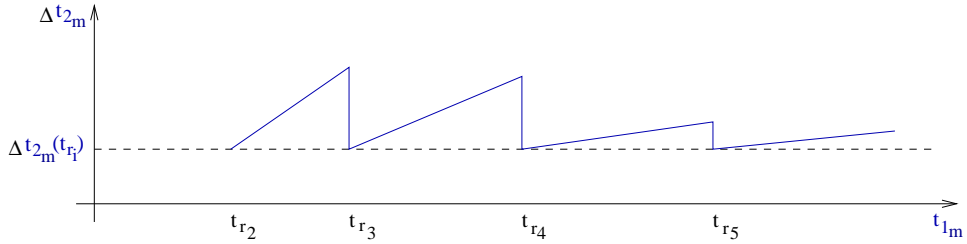


Fig. 7. Evolution of the precision of Δt_{2_m} as more data-points are received.

Equation (30) implies that the uncertainty in computing t_{2_m} increases linearly with the increase of t_{1_m} . Intuitively, this can be observed in Fig. 5: the later the moment t_{1_m} is chosen, the larger Δt_{2_m} gets (i.e., the precision degrades). Since the uncertainty of the relative clock drift Δa_{12} decreases inversely proportionally with Δt_b , the uncertainty in estimating t_{2_m} , Δt_{2_m} returns to an (almost) fixed base-line with each new data point:

$$\Delta t_{2_m}(t_{r_i}) = RTT \left(\frac{2\Delta t_o}{\underline{a_{12}} \underline{a_{12}} \Delta t_b} - \frac{1}{\underline{a_{12}}} \right) \approx const, \quad \forall i \geq 2 \quad (31)$$

(because $\frac{\Delta t_o}{\Delta t_b} \approx 1$, $\underline{a_{12}} \approx 1$, $\overline{a_{12}} \approx 1$).

After each new data-point, the precision on the relative clock drift improves (i.e., Δa_{12} decreases), and, therefore, the slope of the increase of Δt_{2_m} decreases.

3.3.1 Enforcing the Linear Clock Drift Assumption. The assumption of perfectly linear clock drifts (1) only holds for limited time, as a number of factors (temperature, humidity, power source voltage, etc.) may change the oscillator's frequency over time. Thus, it is important to *detect* when assumption (1) no longer holds since tiny-sync assumes linear drifts.

The analysis presented in this section can be used to determine when the linear drift assumption (1) no longer holds: the expected bound on the relative drift Δa_{12} should decrease as $\frac{2RTT}{\Delta t_b}$ (according to (23)). If the linearity assumption on the clock drifts does not hold, Δa_{12} will decrease faster than $\frac{2RTT}{\Delta t_b}$ (and, eventually, if nothing is made to correct the situation, it will become negative as $\overline{a_{12}}$ and $\underline{a_{12}}$ intersect).

The reason for this decrease in Δa_{12} can be understood if one imagines the set of constraints imposed by all data points as a flexible pipe and the two lines that determine $\overline{a_{12}}$, $\underline{a_{12}}$, $\overline{b_{12}}$ and $\underline{b_{12}}$ as two thin rods inside the pipe, spanning the diagonals of the pipe. If the flexible pipe bends (i.e., the clock drift changes), the two rods will get closer to a parallel position, corresponding to a reduction in Δa_{12} (with respect to Δa_{12} for the perfectly straight pipe).

Thus, in practical terms, in the algorithm, after each probe, the computed Δa_{12} is compared with $\frac{2RTT}{\Delta t_b}$, where RTT is the minimum RTT of the stored data-points, and Δt_b is the difference between the t_b timestamps of the stored data-points. If

$$\Delta a_{12} < \frac{2RTT}{\Delta t_b}, \quad (32)$$

the oldest constraint is discarded and a new constraint is acquired. In essence, the algorithm restarts upon detection of the failure of assumption (1).

3.4 Synchronizing an Entire Network

Extending a *pair-wise* synchronization scheme to global network synchronization is relatively straightforward and has been previously explored [Ganeriwal et al. 2003; Maróti et al. 2004]. The simplest approach is to construct a tree using a leader election algorithm [Maróti et al. 2004] and iteratively perform pair-wise synchronization between each parent node and its children. In this section, we will only present the bounds on the performance of multi-hop synchronization.

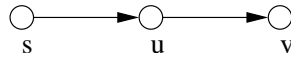


Fig. 8. Synchronization transitivity: if s is synchronized with u , and u is synchronized with v , then s is synchronized with v .

Consider the situation depicted in Fig. 8 where node s synchronizes with node u that, in turn, synchronizes with node v . Node s is able to determine the bounds

$$\underline{a_{su}} \leq a_{su} \leq \overline{a_{su}}, \quad \text{and} \quad (33)$$

$$\underline{b_{su}} \leq b_{su} \leq \overline{b_{su}}, \quad (34)$$

and node u is able to determine the bounds

$$\underline{a_{uv}} \leq a_{uv} \leq \overline{a_{uv}}, \quad \text{and} \quad (35)$$

$$\underline{b_{uv}} \leq b_{uv} \leq \overline{b_{uv}}. \quad (36)$$

If node u sends its bounds $\underline{a_{uv}}$, $\overline{a_{uv}}$, $\underline{b_{uv}}$, and $\overline{b_{uv}}$ to node s , then s can compute the bounds

$$\underline{a_{sv}} \leq a_{sv} \leq \overline{a_{sv}}, \quad \text{and} \quad (37)$$

$$\underline{b_{sv}} \leq b_{sv} \leq \overline{b_{sv}}, \quad (38)$$

where

$$\underline{a_{sv}} = \underline{a_{su}} \underline{a_{uv}}, \quad (39)$$

$$\overline{a_{sv}} = \overline{a_{su}} \overline{a_{uv}}, \quad (40)$$

$$\underline{b_{sv}} = \min \{ \underline{a_{su}} \underline{b_{uv}} + \underline{b_{su}}, \overline{a_{su}} \underline{b_{uv}} + \underline{b_{su}} \}, \quad \text{and} \quad (41)$$

$$\overline{b_{sv}} = \max \{ \overline{a_{su}} \overline{b_{uv}} + \overline{b_{su}}, \underline{a_{su}} \overline{b_{uv}} + \overline{b_{su}} \}. \quad (42)$$

In general, for k nodes in a chain it can be shown that:

$$t_1 = \prod_{i=1}^{k-1} a_{i(i+1)} t_k + \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} (a_{i(i+1)}) b_{i(i+1)} \right\} \quad (43)$$

The corresponding bounds are:

$$\underline{a_{1k}} = \prod_{i=1}^{k-1} \underline{a_{i(i+1)}}, \quad (44)$$

$$\overline{a_{1k}} = \prod_{i=1}^{k-1} \overline{a_{i(i+1)}}, \quad (45)$$

$$\underline{b_{1k}} = \min_{\substack{a_{i(i+1)} \in \{ \underline{a_{i(i+1)}}, \overline{a_{i(i+1)}} \} \\ b_{i(i+1)} \in \{ \underline{b_{i(i+1)}}, \overline{b_{i(i+1)}} \} }} \left\{ \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} (a_{i(i+1)}) b_{i(i+1)} \right\} \right\}, \quad \text{and} \quad (46)$$

$$\overline{b_{1k}} = \max_{\substack{a_{i(i+1)} \in \{ \underline{a_{i(i+1)}}, \overline{a_{i(i+1)}} \} \\ b_{i(i+1)} \in \{ \underline{b_{i(i+1)}}, \overline{b_{i(i+1)}} \} }} \left\{ \sum_{i=1}^{k-1} \left\{ \prod_{j=1}^{i-1} (a_{i(i+1)}) b_{i(i+1)} \right\} \right\}. \quad (47)$$

Since $a_{i(i+1)} \approx \overline{a_{i(i+1)}} \approx 1$ ($\forall i = 1 \dots k-1$), from (46) and (47) it can be inferred that the precision bounds of the offset degrade linearly with the increase in the number of hops (a somewhat intuitive result).

4. EXPERIMENTAL SETUP AND RESULTS

Although the theoretical analysis sheds some light on the expected behavior of the proposed algorithms, several questions can only be answered by an implementation.

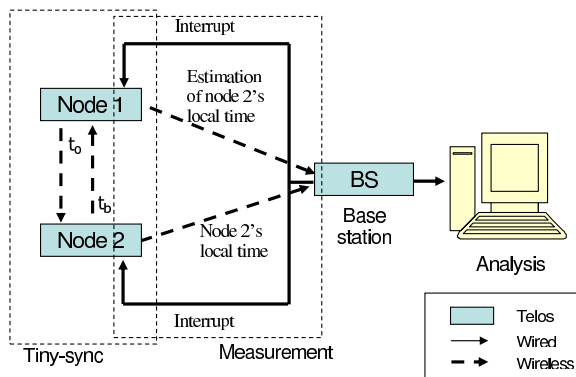


Fig. 9. Experimental setup.

Therefore, we implemented the algorithms in a realistic testbed using state-of-the-art sensor nodes.

4.1 Experimental Setup

In this section, we present the experimental environment we used to measure the performance of the proposed time synchronization protocol.

For reasons explained in Section 3.3, we expect tiny-sync to perform very close to the optimal mini-sync, while using fewer resources. The results presented in Section 4.2.4 confirm our expectation. Thus, in the performance evaluation, we focus on the performance of tiny-sync, which is more likely to be implemented in practice.

We implemented tiny-sync on the Telos platform [Polastre et al. 2005], using TinyOS [Hill et al. 2000] version 1.1.12 (that allows MAC layer time-stamping and byte alignment correction, thus reducing the uncertainties in the delays). In the experiment, we used the external 32kHz crystal clock to obtain current time-stamps. We also implemented tiny-sync on the Mica2 platform [Crossbow] (using the 7.4 MHz internal clock).

4.1.1 Sensor Nodes Configuration. Consistent with the goal of the proposed synchronization protocols, the experimental setup is primarily designed to test the performance of pair-wise synchronization.

Three Telos nodes were used to implement the algorithms and measure the synchronization errors. Figure 9 depicts the experimental setup: nodes 1 and 2 run the tiny-sync algorithm. In our experiment, node 1 estimates the time at node 2. The third node (the base station (BS)) is responsible for querying and collecting clock readings. The base station generates an interrupt periodically (with a period of four seconds). The interrupt is *simultaneously* transmitted to nodes 1 and 2 (through two wires). Upon receiving the interrupt, node 2 sends its local time, and node 1 sends its estimation of the time at node 2 to the base station. After collecting time readings from two nodes, the BS sends them to the host machine for analysis.

4.1.2 Tiny-sync Implementation. Figure 10 illustrates the moments we record the four time-stamps in Section 3.1.1. We used the hooks provided by TinyOS to

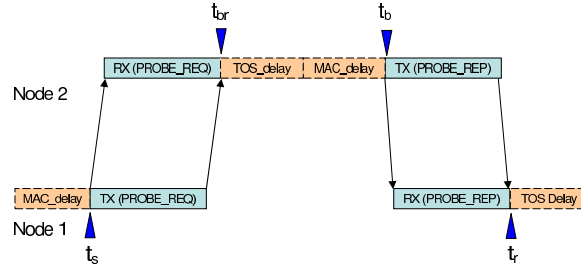


Fig. 10. Time-stamping in tiny-sync.

time-stamp the packet just before the first byte of the packet is transmitted and immediately after the last byte of the packet is received. Thus, all of the non-deterministic delays (*RTT* in Section 3.3) are due to the transceiver and scheduling of TinyOS. Our measurements showed that the one-way delay is approximately 1.419 ms (2.838 ms round-trip time).

4.1.3 Synchronization Errors Measurements. We define the time synchronization error as the time difference between the real clock reading of a sensor node and the estimated clock value by the other sensor node. There are two distinct sources of errors when measuring the synchronization error:

- the synchronization algorithm and
- the measurement procedure itself.

As previously mentioned, in the experimental setup, the base station triggers simultaneous interrupts at the two nodes running tiny-sync. This measurement method eliminates the radio transceiver delay uncertainties from the measurement process. However, it introduces other sources of non-deterministic delays that may result in erroneous measurements. Figure 11 shows the possible sources of delays affecting the measurements. When an interrupt occurs, unless interrupts are suspended, the processor first completes the current executing instruction, saves the program counter (PC) and the status register (SR), then jumps to the corresponding interrupt service routine (ISR). In our case, the ISR reads the current clock. Unfortunately, if the processor is in the middle of a non-interruptible operation at the time the interrupt arrives (i.e., the interrupts were suspended), or there are other pending interrupts, the measurement interrupt will be delayed for an unknown time (in our experiments, we found it to be up to $61\ \mu\text{s}$).

There can be several ways to define the measurement error. Assume we measure the synchronization error for each interrupt j , where $j = 1, 2, 3, \dots$. We define the measurement error E_j of the synchronization error between node n_1 and n_2 at measurement j as

$$E_j = x_{n_1,j} - x_{n_1,j-1} - \widehat{a}_{12}(x_{n_2,j} - x_{n_2,j-1}), \quad (48)$$

where node n_1 estimates the clock of node n_2 , and $x_{n_i,j}$ is the time-stamp of node n_i at measurement j , and \widehat{a}_{12} is defined in (9). That is, the measurement error is the inter-measurement time difference between participating nodes. The measurement errors can be reduced by discarding the data-points whose measurement errors are

greater than a threshold value τ . In this experiment, we set the parameter τ to $33\mu s$ (i.e., we discarded all measurements with an error higher than $33\mu s$). We observed that 94.42% of the measurements have their errors of smaller than the threshold value.

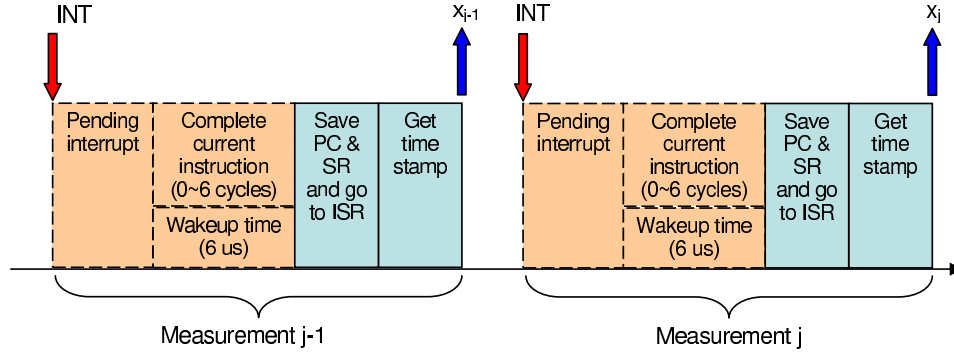


Fig. 11. Sources of delay in the measurement process error.

4.1.4 Global Synchronization. In order to evaluate global synchronization errors, we added three more Telos motes to the existing two, thus forming a chain of five nodes. Each node $i + 1$ estimates the clock of its parent (node i). The query node periodically generates a measurement interrupt. Upon the receipt of each interrupt, each node i reports the value of its local clock as well as its estimate of the clock at node $i - 1$ (obviously, with the exception of node 1 that only sends its local clock). The multihop configuration was enforced by static routing.

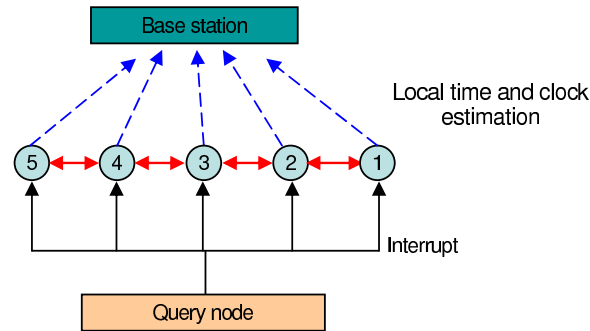


Fig. 12. Experimental setup for global synchronization.

4.2 Experimental Results

In this section, we present the results of the experiments. We focused on the performance of tiny-sync as well as on comparing its performance with mini-sync and other existing time synchronization protocols.

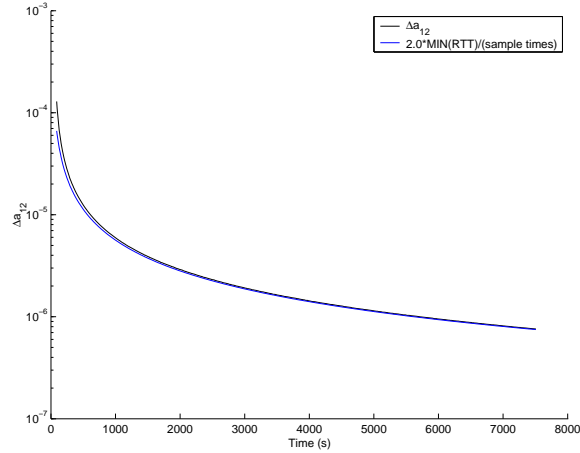


Fig. 13. The evolution of the bound on the relative drift Δa_{12} and the predicted evolution (23) for the first experiment.

4.2.1 *Time Synchronization Error of Tiny-sync.* Using the setup described in Section 4.1, we ran the first experiment for 2.1 hours. Figure 13 shows the bound on the relative drift of the two clocks Δa_{12} and the theoretical bound $\frac{2RTT}{\Delta t_b}$ in (23). It is clear that the relation (23) holds exceedingly well, implying that the assumption of linear drift (1) holds within the bounds of our accuracy for this first experiment. In Section 4.2.3, we present the results of another experiment where this assumption is purposely broken.

Figure 14 shows the synchronization error of tiny-sync as a function of time. The average synchronization error achieved by tiny-sync is $12.075 \mu s$. The synchronization error is smaller than the quantization error: for the 32 kHz external clock oscillator, each unit of time is equal to $31.21 \mu s$.

The algorithm converges quickly to the nominal precision: it achieves the average precision with the first few probes. Considering that the entire network can be synchronized as fast as a pair of nodes (Section 3.4), only a few seconds are necessary to synchronize an entire network.

Figure 15 shows the distribution of the synchronization errors. The graph indicates that most (about 99.48%) of the time synchronization errors are smaller than $30 \mu s$. Thus, tiny-sync is suitable for applications that require good accuracy and consistency.

Figure 16 shows the synchronization errors of tiny-sync algorithm as a function of the number of hops as well as the synchronization interval. The synchronization interval is the time between two consecutive probe messages. The synchronization error increases both with the number of hops between two nodes as well as with the synchronization interval.

4.2.2 *Comparison with Other Approaches.* In this section, we compare the performance of tiny-sync with two other time synchronization approaches for WSN: TPSN [Ganerival et al. 2003] and FTSP [Maróti et al. 2004] as well as a simple linear fit on the collected data. In order to make the comparison as fair as possible,

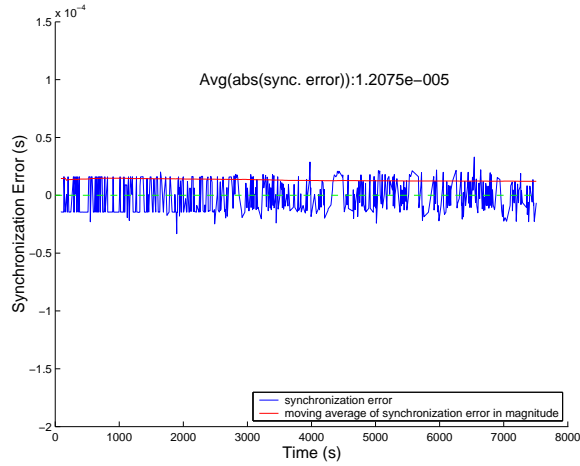


Fig. 14. Tiny-sync pairwise synchronization error.

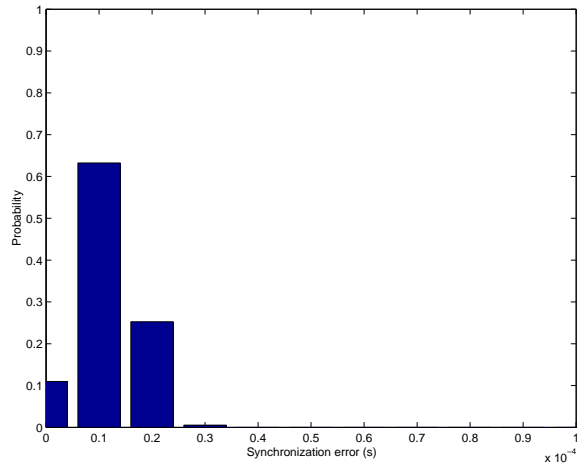


Fig. 15. Distribution of the synchronization errors.

we did not filter out the measurement errors for any synchronization protocol (i.e., all measurement data were included in comparison).

TPSN uses a similar probing method as tiny-sync to estimate the clock of the target sensor node. It also exploits the MAC layer time-stamping to reduce the timing uncertainties. Thus, we used the same data collection to determine the time synchronization error of tiny-sync and TPSN. For FTSP, we downloaded the latest version of FTSP source code from [Maróti et al. 2005] and implemented it on the same Telos motes. We used a test program provided by FTSP to evaluate time synchronization errors of the sensor nodes running FTSP.

For the linear fit we used least square to minimize the error between a linear fit and the data points (both (t_o, t_b) and (t_b, t_r)). We also tested the performance

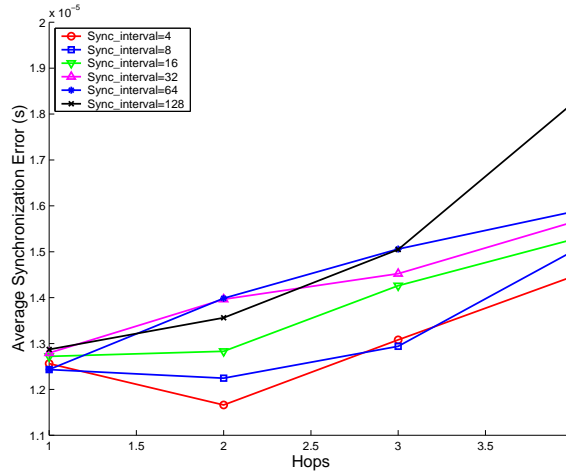


Fig. 16. Tiny-sync global synchronization error.

of a one-way linear fit (that only considers forward probes (t_o, t_b)) and the results were nearly identical. However, while a one-way linear fit only requires one probe packet, it also requires hardware-dependent calibration. Thus, in what follows, we present the results of the two-way fit (again, practically identical with the one-way fit), as the two way fit match the properties of the proposed scheme (in terms of need for calibration).

Since the results of the linear fit are sensitive to the number of data points considered, we plotted synchronization error of the linear fit as a function of the number of data points considered for the fit. The results are shown in Fig. 17. The graph shows that the linear fit has an optimum number of points - both a lower and a higher number of points lead to a decrease in accuracy. Intuitively, if only a small number of points are used, the accuracy of the linear fit is limited by the quality of the data provided by these points. However, if the drift is non-linear, as the number of data-points considered for the linear fit increases, the fit necessarily becomes an increasingly poor approximation of the current relationship between the two clocks (as it minimizes the error to *all* considered points). In the following comparisons we use both 128 points that optimizes the performance of the fit for our experimental setup as well as a 2 point fit that has a comparable computational and storage complexity with tiny-sync.

Figure 18 shows the comparison of the synchronization errors as a function of the number of hops between sensor nodes. When the synchronization interval is 4 s, tiny-sync and TPSN show similar results. However, when the synchronization interval is 128 s, tiny-sync clearly outperforms TPSN and FTSP. Furthermore, the synchronization error of FTSP increases faster than tiny-sync and TPSN with the number of hops. The linear fit that uses 128 points (optimal for this data-set) at four second synchronization interval, slightly outperforms tiny-sync for all number of hops. This slight improvement is attained at a price of higher computational and storage resources as well as at a loss of generality: the linear fit has to be optimized for the particular degree of the non-linearity of the clock drift. When only two

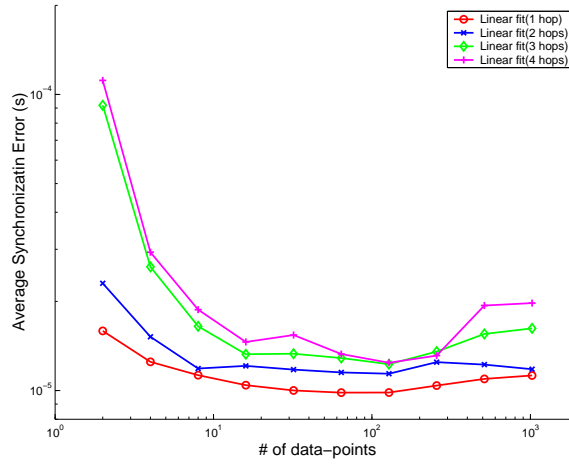


Fig. 17. Synchronization errors as a function of the number of points considered for the linear fit and tiny-sync.

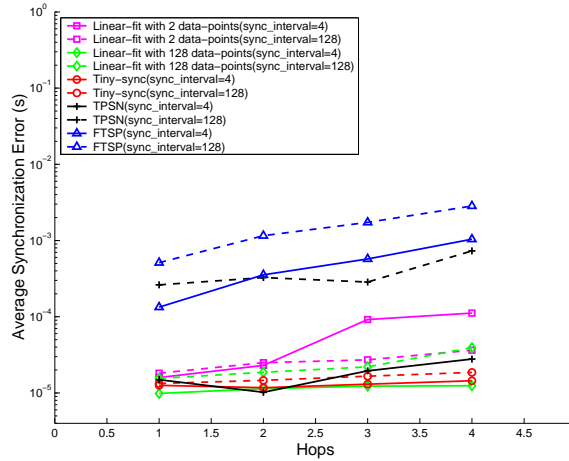


Fig. 18. Comparison of synchronization errors as a function of the number of hops.

points are used for the linear fit or when the synchronization interval is higher (128 seconds), tiny-sync outperforms the linear fit.

Figure 19 shows the comparison of the synchronization errors as a function of the synchronization intervals. Tiny-sync outperforms TPSN and FTSP in terms of synchronization error, and it shows reduced variation with the increase in the synchronization interval. Synchronization error of TPSN increases faster than others as the synchronization interval increases because TPSN does not compensate for the clock drift. The synchronization error of FTSP increases slowly as the number of hops increases, as FTSP uses the linear regression to compensate for the drift. The accuracy of the linear fit decreases slightly with the increase in the synchronization interval, as the number of points is also correspondingly reduced. The 128

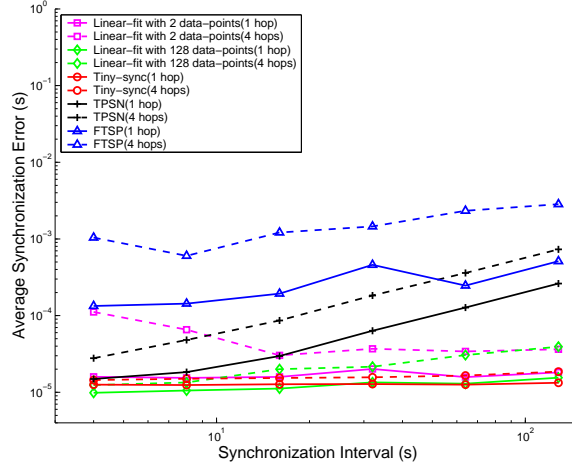


Fig. 19. Comparison of synchronization errors as a function of the synchronization interval.

points linear fit outperforms all synchronization algorithms for a single hop, and a small synchronization interval.

4.2.3 Robustness Evaluation. In this section, we present results of the experiment that tested the robustness of tiny-sync with respect to large variations in the relative drift of the clocks. Essentially, we break the assumption (1) and study its effect on the precision of the algorithm. To induce a variation in the drift, we drastically changed the temperature of node 2, while keeping node 1 at a constant room temperature. One hour into the experiment, we placed node 2 into a box filled with ice and closed the lid. One hour later, we removed the node from the ice box and continued the experiment for a total duration of 4.48 hours.

The change in temperature, as expected, negatively affected the bounds on the relative drift of the clocks (Figure 20(a)). We used the procedure detailed in Section 3.3.1 to detect the change in the drift of the clocks and restart the algorithm. Figure 20(b) shows the moments when the algorithm detects the change in the relative drift of the clocks and restarts.

Figure 21 shows the synchronization error for the second experiment. Despite the significant change in the relative drift while node 2 was in the ice box, the average synchronization error during that period did not change. The average precision of this experiment ($10.78 \mu s$) was actually better than that of the initial experiment ($12.07 \mu s$ in Fig. 14). This experiment validates that the restart detection algorithm shows that tiny-sync is robust with respect to changes in the clock drifts.

Figure 22 shows the results of the robustness of tiny-sync with respect to packet loss. In this experiment, we simulated the environment in which synchronization packets experience random loss. Using the data collected from the first experiment, we varied the packet loss rate (for both the probe request and reply) from 0 to 90%. Except for cases with large packet loss rate and large synchronization intervals, the precision was not significantly affected.

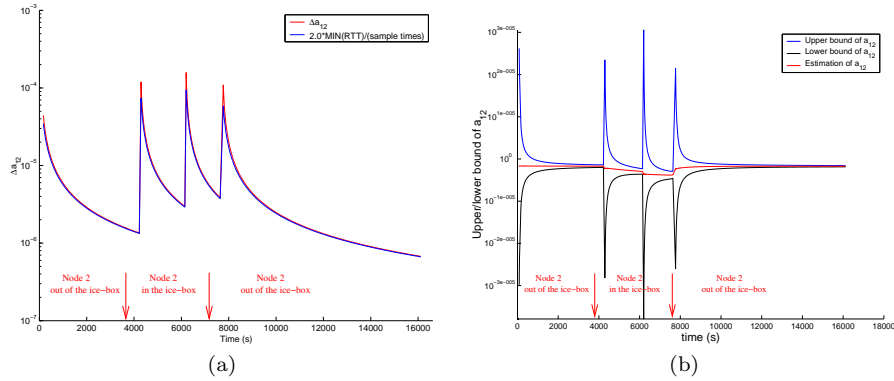


Fig. 20. Behavior of tiny-sync when the clocks are forced to change their relative clock drift by sudden changes in the temperature of one of them (by temporary placing it in an ice-box). During the cool-down and warm-up period, tiny-sync restarts several times. (a) the evolution of the bound on the relative drift Δa_{12} and the predicted evolution (23); (b) the evolution of the bounds on the drift.

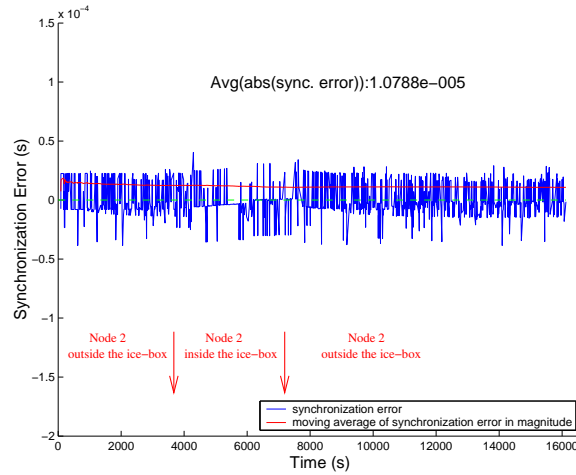


Fig. 21. The synchronization error when nodes were subjected to variation of the relative drift.

4.2.4 *Comparison of Tiny-sync and Mini-sync.* In this section, we compare the performance of tiny-sync and mini-sync, which is optimal (Section 3.2). Figures 23 and 24 show the synchronization errors of tiny-sync and mini-sync as a function of the number of hops and synchronization interval, respectively. As expected, there is no significant difference in performance between tiny-sync and mini-sync. Given the fact that tiny-sync requires less resources (computational and storage) than mini-sync, this result suggests that using tiny-sync is preferred to mini-sync.

Although one would expect mini-sync to always outperform tiny-sync, the results show that, in practice, sometimes tiny-sync can show better precision than mini-sync. It turns out that the selection of the first data-points to be used in the drift estimation can affect the overall performance of the algorithm. Details on this effect

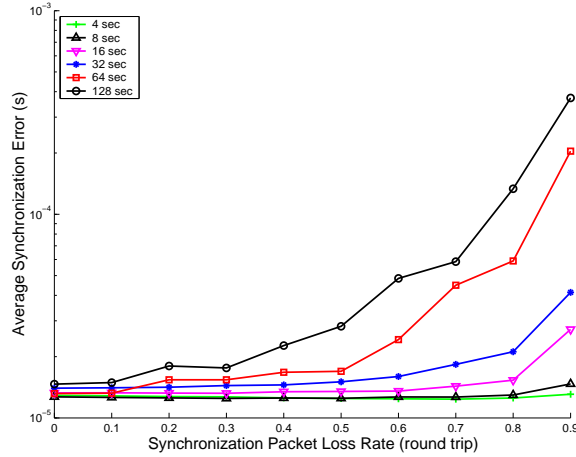


Fig. 22. The synchronization error with packet loss.

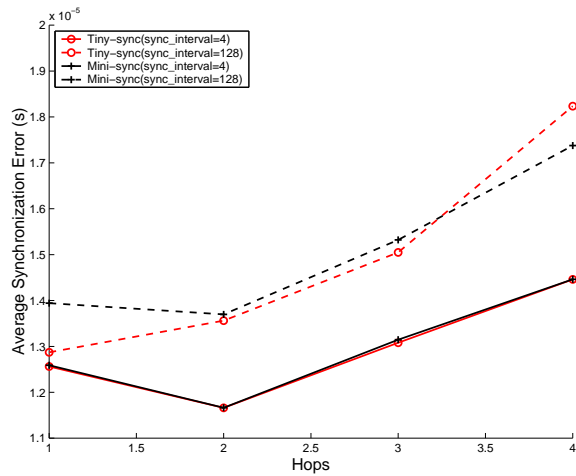


Fig. 23. Synchronization error of tiny-sync and mini-sync as a function of the number of hops.

are presented in Section 4.2.6.

4.2.5 Experimental Results on Mica2 Platform. We also implemented tiny-sync on the Mica2 platform [Crossbow] using TinyOS. In this experiment, we used the far more accurate 7.4 MHz internal clock to obtain time-stamps. The average pair-wise synchronization error achieved by tiny-sync on Mica2 was $1.3868 \mu\text{s}$. According to published data [Maróti et al. 2004; Ganeriwal et al. 2003], the pair-wise accuracies of RBS, TPSN and FTSP on the Mica platform are $7 \mu\text{s}$, $16.9 \mu\text{s}$ and $1.48 \mu\text{s}$, respectively. The seemingly poor result for TPSN is for a single time-stamp exchange on a Mica 1 platform (that has a slower clock and radio transceiver). We expect that an averaging implementation on the Mica2 platform would result in a precision similar to FTSP and tiny-sync. The accuracy of tiny-sync is very

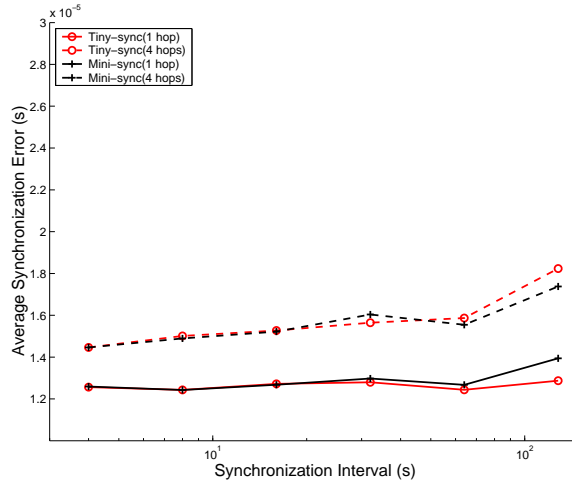


Fig. 24. Synchronization error of tiny-sync and mini-sync as a function of the synchronization interval.

close to the accuracy of FTSP (tiny-sync is slightly better). It is likely that both approaches reached the lower bound of achievable accuracy on the Mica2 platform (due to non-deterministic delays, most likely introduced by the transceiver and/or clock inaccuracies).

Figure 25 shows the pair-wise synchronization errors for tiny-sync using Telos and Mica2 motes. In this experiment, we varied the synchronization interval from 4 s to 1024 s. When the synchronization interval is short, the synchronization error on the Mica2 platform is better than that on Telos due to the clock resolution on the Mica2 platform; however, the external crystal clock of the Telos platform is more stable than the internal clock of Mica2 in the long run. Thus, for short synchronization intervals, the synchronization errors on Mica2 are smaller than on the Telos motes; however, the situation is reversed when the synchronization interval is longer than 128 s.

4.2.6 Tiny-sync Initialization. As mentioned in Section 4.2.4, the selection of data-points in the beginning of the algorithm may affect the synchronization precision in the initial phase (especially for long synchronization intervals).

Figure 26 shows an example of a data set with high synchronization error for the beginning of the experiment. The synchronization interval is 128s. The reason for the poor performance is in the choice of the initial data-points. If the first few data samples are of poor quality (i.e., with large non-deterministic delays), the initial precision will suffer (until at least two reasonably “good” data-points are received). A simple method of forcing tiny-sync to start with “good” data-points is to start with the best two data-points out of the first N . Figure 27 shows the synchronization error when N is 10.

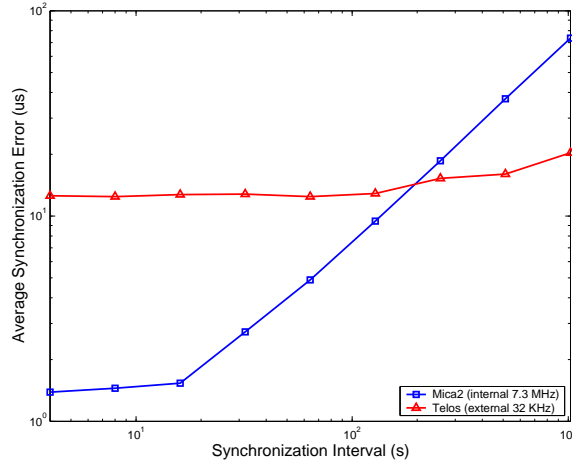


Fig. 25. Synchronization error of tiny-sync on Mica2 and Telos platforms.

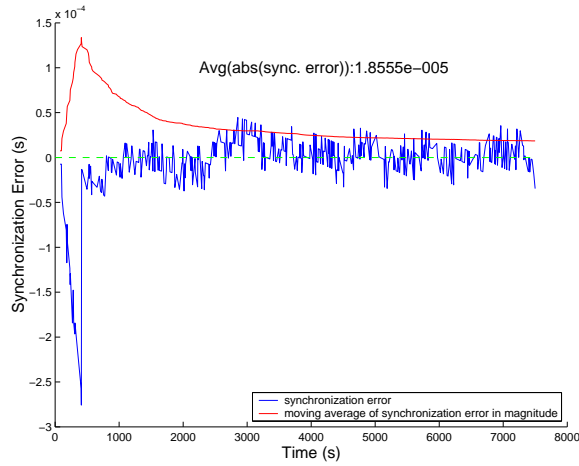


Fig. 26. Synchronization errors of tiny-sync without poor initial data.

5. ACKNOWLEDGMENTS

We would like to sincerely thank the anonymous reviewers. Their comments truly improved this paper from more than one standpoint. We also thank NSF and the Center for Advanced Computing and Communication at NCSU for funding this work.

6. CONCLUSION

In this paper, we proposed and evaluated the performance of two simple time synchronization algorithms suitable for wireless sensor networks. The algorithms perform pair-wise synchronization and can be used as the basic building block for synchronizing an entire network. While the performance of the two algorithms may

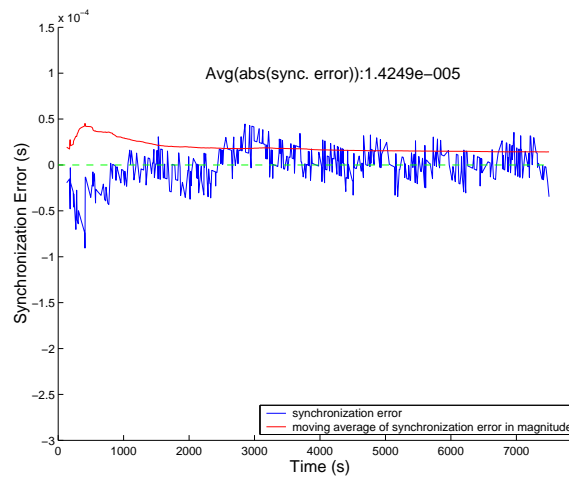


Fig. 27. Synchronization errors of tiny-sync when the best two out of the first ten data-points are used to initialize the algorithm.

vary in theory, in practice, they yield very similar results. Thus, the simplest one, tiny-sync, is likely to be practically implemented. The main advantage of tiny-sync is its simplicity and parsimonious resources requirements. Other advantages include robustness to large variations in clock drift and its ability to achieve fast synchronization of an entire network. Experimental results show that it performs as well or better than existing time synchronization approaches for wireless sensor networks.

REFERENCES

- AKYILDIZ, I., SU, W., SANKARASUBRAMANIAM, Y., AND CAYIRCI, E. 2002a. A survey on sensor networks. *IEEE Communication Magazine* 40, 8 (Aug.), 102–116.
- AKYILDIZ, I., SU, W., SANKARASUBRAMANIAM, Y., AND CAYIRCI, E. 2002b. Wireless sensor networks: A survey. *IEEE Computer* 38, 4 (Mar.), 393–422.
- ASADA, G., DONG, T., LIN, F., POTTIE, G., KAISER, W., AND MARCY, H. 1998. Wireless integrated network sensors: Low power systems on a chip. In *Proc. of the 24th European Solid-State Circuits Conference*. The Hague, Netherlands.
- CRISTIAN, F. 1989. Probabilistic clock synchronization. *Distributed Computing* 3, 3, 146–158.
- CROSSBOW. Mica2 wireless measurement system. <http://www.xbow.com/Products/productsdetails.aspx?sid=72>.
- DAI, H. AND HAN, R. 2004. Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review* 8, 1, 125–139.
- ELSON, J. AND ESTRIN, D. 2001. Time synchronization for wireless sensor networks. In *Proc. of the 2001 International Parallel and Distributed Processing Symposium (IPDPS), Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*. San Francisco, CA.
- ELSON, J., GIROD, L., AND ESTRIN, D. 2002. Fine-grained network time synchronization using reference broadcasts. In *UCLA Technical Report 020008*.
- GANERIWAL, S., KUMAR, R., AND SRIVASTAVA, M. B. 2003. Timing-sync protocol for sensor networks. In *Proc. of the First ACM Conference on Embedded Networked Sensor System (SenSys)*. 138–149.

- GIROD, L. D. 2005. A self-calibrating system of distributed acoustic arrays. Ph.D. thesis, University of California Los Angeles.
- GREUNEN, J. V. AND RABAEY, J. 2003. Lightweight time synchronization for sensor networks. In *WSNA '03*.
- HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D. E., AND PISTER, K. S. J. 2000. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*. 93–104.
- HOFMANN-WELLENHOF, B., LICHTENEGGER, H., AND COLLINS, J. 1997. *Global Positioning System: Theory and Practice*, 4th ed. Springer-Verlag.
- KARP, R., ELSON, J., ESTRIN, D., AND SHENKER, S. 2003. Optimal and global time synchronization in sensor networks. Tech. Rep. 0012, CENS.
- LEMMON, M., GANGULY, J., AND XIA, L. 2000. Model-based clock synchronization in networks with drifting clocks. In *Proc. of the 2000 Pacific Rim International Symposium on Dependable Computing*. Los Angeles, CA, 177–185.
- LI, Q. AND RUS, D. 2004. Global clock synchronization in sensor network. In *INFOCOM*.
- LU, G., SADAGOPAN, N., KRISHNAMACHARI, B., AND GOEL, A. 2005. Delay efficient sleep scheduling in wireless sensor networks. In *Proc. of Infocom 2005*.
- MARÓTI, M., KUSY, B., SIMON, G., AND LÉDECZI, A. 2004. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press, 39–49.
- MARÓTI, M., KUSY, B., SIMON, G., AND LÉDECZI, A. 2005. Implementation of flooding time synchronization protocol. <http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/contrib/vu/>.
- MEIER, L., BLUM, P., AND THIELE, L. 2004. Internal synchronization of drift-constraint clocks in ad-hoc sensor networks. In *MobiHoc*.
- MILLS, D. L. 1991. Internet time synchronization: the network time protocol. *IEEE Trans. Communications* 39, 10 (Oct.), 1482–1493.
- MILLS, D. L. 1994. Improved algorithms for synchronizing computer network clocks. In *Proc. of ACM Conference on Communication Architectures (ACM SIGCOMM'94)*. London, UK.
- PALCHAUDHURI, S., SAHA, A., AND JOHNSON, D. B. 2004. Adaptive clock synchronization in sensor networks. In *ISPN '04: Proceeding of The Third International Symposium on Information Processing in Sensor Networks*. 340–348.
- PING, S. 2003. Delay measurement time synchronization for wireless sensor networks. In Intel Research, IRB-TR-03-013.
- POLASTRE, J., SZEWCZYK, R., AND CULLER, D. 2005. Telos: Enabling ultra-low power wireless research. In *In Proceedings of IPSN/SPOTS*. Los Angeles, CA.
- RÖMER, K. 2001. Time synchronization in ad hoc networks. In *Proc. of ACM Mobihoc*. Long Beach, CA.
- SIMON, G., MARÓTI, M., LÉDECZI, A., BALOGH, G., KUSY, B., NÁDAS, A., PAP, G., SALLAI, J., AND FRAMPTON, K. 2004. Sensor network-based countersniper system. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press.
- SIVRIKAYA, F. AND YENER, B. 2004. Time synchronization in sensor networks: A survey. *IEEE Network* 18, 45–50.
- SU, W. AND AKYILDIZ, I. F. 2005. Time-diffusion synchronization protocol for wireless sensor networks. *IEEE/ACM Transactions on Networking* 13, 384–397.
- SUNDARARAMAN, B., BUY, U., AND KSHEMKALYANI, A. D. 2005. Clock synchronization in wireless sensor networks: A survey. *Ad-Hoc Networks* 3, 281–323.
- XU, N., RANGWALA, S., CHINTALAPUDI, K. K., GANESAN, D., BROAD, A., GOVINDAN, R., AND ESTRIN, D. 2004. A wireless sensor network for structural monitoring. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press, 13–24.

A. PROOF OF THEOREM 1

In this section we provide the proof for Theorem 1. Let us denote with (A, B) the line determined by the points A and B and with $m(A, B)$ the slope of the line (A, B) . We start by proving the following lemma:

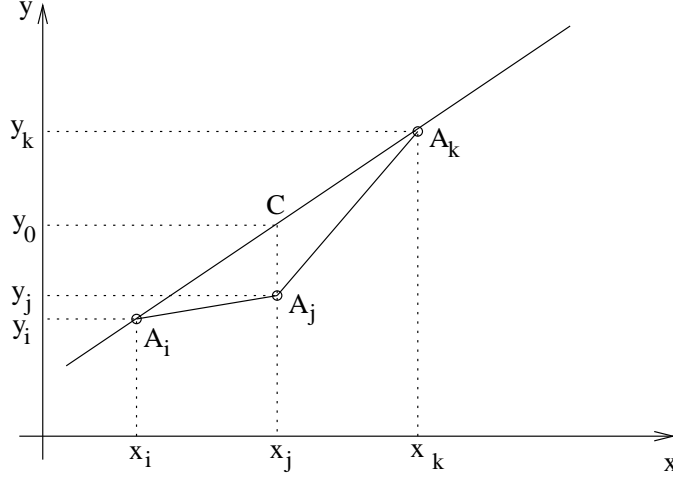


Fig. 28. Illustration for Lemma 2.

Lemma 2 Consider three points A_i, A_j, A_k of coordinates (x_i, y_i) , (x_j, y_j) and (x_k, y_k) respectively with $x_i < x_j < x_k$. Then

$$m(A_i, A_j) < m(A_i, A_k) < m(A_j, A_k), \quad (49)$$

if and only if A_j is below the line (A_i, A_k) . Similarly,

$$m(A_i, A_j) > m(A_i, A_k) > m(A_j, A_k), \quad (50)$$

if and only if A_j is above the line (A_i, A_k) .

Proof Denote with y_0 the y-coordinate of point C where the vertical line from A_j intersects the line (A_i, A_k) :

$$y_0 = \frac{(y_k - y_i)x_j + y_i x_k - y_k x_i}{x_k - x_i}. \quad (51)$$

The expression “ A_j is below the line (A_i, A_k) ” is shorthand for $y_j < y_0$. With this notation, if and only if $y_j < y_0$, then

$$m(A_i, A_j) = \frac{y_j - y_i}{x_j - x_i} < \frac{y_0 - y_i}{x_j - x_i} = m(A_i, A_k) = \frac{y_k - y_0}{x_k - x_j} < \frac{y_k - y_j}{x_k - x_j} = m(A_j, A_k), \quad (52)$$

which is (49). By a similar argument it can be shown that (50) holds if and only if $y_j > y_0$.

Theorem 1 Any constraint A_j which satisfies

$$m(A_i, A_j) \leq m(A_j, A_k) \quad (53)$$

for at least one set of integers $1 \leq i < j < k$ can be safely discarded as it will never constrain the bounds \underline{a}_{12} , \overline{a}_{12} , \underline{b}_{12} and \overline{b}_{12} more than any existing constraints.

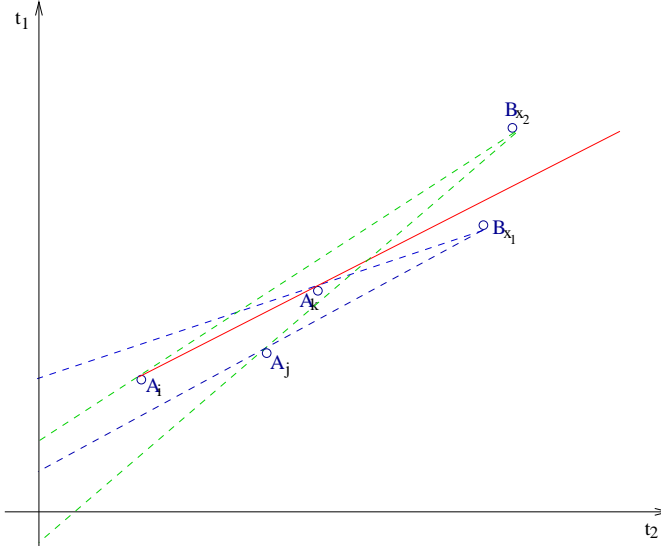


Fig. 29. A_j can be safely discarded as A_i or A_k will result in better estimates in all cases.

Proof

Consider the situation depicted in Fig. 29. The constraints A_i , A_j and A_k satisfy condition (53). Consider a constraint B_x of coordinates (t_{2_x}, t_{1_x}) such that $t_{1_x} > t_{1_k}$ and $t_{2_x} > t_{2_k}$. Constraint B_x and one of the constraints A_i , A_j or A_k may determine the upper bound \overline{a}_{12} . Denote with $\overline{a}_{12}(A_j, B_x)$ the upper bound on a_{12} determined by the constraints A_j and B_x . We can distinguish two cases:

1. When B_x is *below* the line (A_i, A_k) (e.g., position B_{x_1} in Fig. 29). In this case, because both A_j and B_{x_1} are under the line (A_i, A_k) , A_k is above the line (A_j, B_{x_1}) . From lemma 2 it follows that

$$m(A_k, B_{x_1}) < m(A_j, B_{x_1}), \quad (54)$$

which is equivalent with

$$\overline{a}_{12}(A_k, B_{x_1}) < \overline{a}_{12}(A_j, B_{x_1}). \quad (55)$$

Thus, the constraint (A_k, B_x) is tighter than the constraint (A_j, B_x) for the case when B_x is below the line (A_i, A_k) .

2. When B_x is *above* the line (A_i, A_k) (position B_{x_2} in Fig. 29). In this case, by a similar logic, it can be shown that

$$\overline{a}_{12}(A_i, B_{x_2}) < \overline{a}_{12}(A_j, B_{x_2}), \quad (56)$$

i.e., the constraint (A_i, B_x) is tighter than the constraint (A_j, B_x) for the case when B_x is above the line (A_i, A_k)

Thus, in both cases, one of the upper bounds $\overline{a_{12}}(A_i, B_{x_2})$ or $\overline{a_{12}}(A_k, B_{x_1})$ is tighter than the upper bound introduced by A_j , $\overline{a_{12}}(A_j, B_x)$, showing that constraint A_j will always results in a looser upper bound than A_i or A_k , and, thus, enabling its disposal without any possible reduction in optimality of the solution.

Received August 2005;