Simple, Accurate Time Synchronization for Wireless Sensor Networks

Mihail L. Sichitiu and Chanchai Veerarittiphan
Electrical and Computer Engineering Department
North Carolina State University
Raleigh, NC 27695-7911
Email: {mlsichit,cveerar}@eos.ncsu.edu

***Abstract*— Time synchronization is important for any distributed system. In particular, wireless sensor networks make extensive use of synchronized time in many contexts (e.g. for data fusion, TDMA schedules, synchronized sleep periods, etc.). Existing time synchronization methods were not designed with wireless sensor networks in mind, and need to be extended or redesigned. Our solution centers around the development of a deterministic time synchronization method relevant for wireless sensor networks. The proposed solution features minimal complexity in network bandwidth, storage and processing and can achieve good accuracy. Highly relevant for sensor networks, it also provides tight, deterministic bounds on both the offsets and clock drifts. A method to synchronize the entire network in preparation for data fusion is presented. A real implementation of a wireless ad-hoc network is used to evaluate the performance of the proposed approach.**

Keywords: synchronization, wireless sensor networks, model based, clock drift bound.

I. Introduction

The availability of small, cheap microsensors and low power wireless communications nowadays enables the deployment of large arrays of wireless sensor networks allowing us to monitor and, eventually, control many aspects of the physical world. Applications include tagging small animals unobtrusively, environmental monitoring, tagging small objects in a hospital or factory warehouse. In tactical environments, it can be used to track the movements of troops or targets.

The sensed data is of limited usage if it is not accompanied by the coordinates of the sensor - position and time stamp. This is perhaps the primary reason for clock synchronization in wireless sensor networks. But there are also other vital functions which depend on clock synchronization.

A basic function of a wireless sensor network is data fusion, i.e. combining data from multiple sensors into high level data. For example, a vehicle going through a sensor network equipped with acoustic sensors can be detected by different sensor nodes at different moments (corresponding to the moments when the vehicle entered the detection range of those nodes). A fusion node receiving the raw information from the sensor nodes can refine it by estimating the speed and the direction of the sensed vehicle. For this application (and most other applications), synchronized timestamps (together with position information) are essential.

Sensor networks are expected to have very small form factors and be cheap such that they can be deployed in very large numbers. This precludes spending a large amount of resources on a large, expensive power source for each node. Once deployed, the sensor networks are usually unattended, so battery replacement is out of the question – the life of the sensor network is equal to the life of its batteries. Finally, such a network is typically expected to work for extended periods of time (weeks, months, and in some cases, years). There is no better way to conserve energy but to put the nodes to sleep (using low power components only goes so far). However, to perform its function the network should wake up at periodic intervals. It is essential that all the nodes are able to wake up at the *same time* to be able to exchange information.

In addition, various time division multiple access (TDMA) schemes proposed in literature for ad hoc networks assume clock synchronization of the nodes [1].

Nodes could be equipped with a global positioning system (GPS) [2] to synchronize them, but currently this is a costly (in size, cost and power consumption) solution.

There is a considerable amount of work available in the field synchronization for distributed systems [3]–[14]. However most of the existing methods do not take into account the limited resources available for sensor networks and require either error free operation, significant storage for the data samples, or processing power not plentiful in a sensor network.

Elson and Estrin presented an interesting technique called post-facto synchronization [15] which is also based on unsynchronized local clocks but limits synchronization to the transmission range of the mobile computing nodes. The precision achieved by their approach is very good, but lacks a deterministic bound on the clock drift.

Recently, other approaches suitable for wireless sensor networks synchronization were published [16] or are under review [17]. However, those approaches are still comparatively computation intensive, and do not provide firm bounds on the achieved precision.

This paper presents a synchronization algorithm for drifting clocks similar to that found in [4], [18] but processes the collected data in a different manner. The proposed algorithm preserves the fault-tolerance properties of the methods in [4], [18].

Due to unpredictability and imperfect measurability of message delays in a networked environment, physical clock synchronization is *always imperfect*. We will present two closely related algorithms suitable for synchronizing two wireless nodes. We will also present a scheme building on these algorithms capable to synchronize 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 resources and very limited resources respectively. The two algorithms feature:

- *Drift awareness:* the algorithm not only takes the drift of the clock into account, but also finds tight bounds on the drift which allows for corrections.
- *Tight bounds on the precision:* Most other algorithms provide best estimates for the offset and the drift of the clocks, and possibly probabilistic bounds on these estimates. Our approach deliverers tight, *deterministic* bounds on these estimates, such that absolute information can be deduced about ordering and simultaneous events.
- *Accuracy:* given small uncertainty bounds on the delays the 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.
- *Insensitivity to communication errors:* wireless communications are notoriously error prone, and thus one cannot rely on receiving correctly all messages. The presented approach works correctly even if a large percentage of the messages are lost.

The main contribution of the paper is the development of a *simple* algorithm which delivers accurate offset and drift information together with *tight, deterministic* bounds on them. This is a highly desirable property for wireless sensor networks. Assume that two nodes, a transmitter and a receiver are in sleep mode. To ensure that the receiver wakes up before the transmitter, the offset error *and* the drift error between their clocks have to be known.

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 bi-directional data transmission. However, the algorithms provide very good precision (micro-second if crafted carefully) and bounds on the precision while using very little resources and thus being especially well suited for wireless sensor networks.

II. Data Collection Algorithms

In this Section a simple data collection algorithm and possible enhancements are presented.

A. A simple data collection algorithm

We will use a classical data collection algorithm [4], [8], [11], [16]. We will however process the data stream differently. Consider two wireless nodes 1 and 2, with their hardware clocks $t_1(t)$ and $t_2(t)$ respectively, 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 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) can be approximated with good accuracy by an oscillator with fixed frequency:

$$t_i(t) = a_i t + b_i, \tag{1}$$

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.

From (1) it follows that t_1 and t_2 are linearly related:

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

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 timestamped right before it is sent with t_o. Upon receipt, node 2 timestamps the probe t_b and returns it immediately (we will shortly relax this requirement) to node 1 which timestamps it upon receipt t_r. Fig. 1 depicts such an exchange.

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:

$$t_o(t) < a_{12} t_b(t) + b_{12}, \tag{3}$$

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

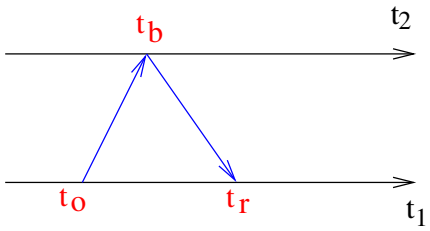


Fig. 1. A probe message from node 1 is immediately returned by node 2 and timestamped at each send/receive point resulting in the data-point (t_o, t_b, t_r).

The measurement described above will be repeated several times and each probe which returns will provide a new data point and thus new constraints on the admissible values of a_{12} and b_{12}.

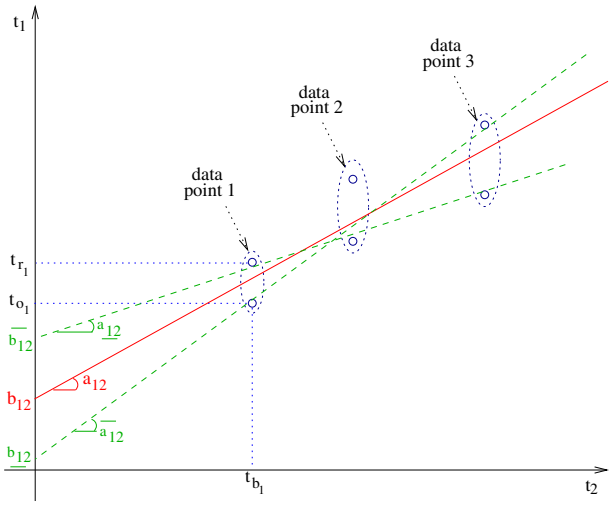


Fig. 2. The linear dependence (2) and the constraints imposed on a_{12} and b_{12} by three data-points.

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 be represented by two constraints in the system of coordinates given by the local clocks of the two nodes t_2 and t_1. One constraint is (t_b, t_o) and the other one is (t_b, t_r).

Inequalities (3) and (4) graphically require that the line representing the relationship (2) is positioned *between* the two constraints of each data-point. The exact values of a_{12} and b_{12} cannot be exactly 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:

$$\underline{a_{12}} \leq \quad a_{12} \quad \leq \overline{a_{12}}, \tag{5}$$

$$\underline{b_{12}} \leq \quad b_{12} \quad \leq \overline{b_{12}}. \tag{6}$$

Not all combinations of a_{12} and b_{12} satisfying (5) and (6) are valid, but all valid combinations satisfy (5) and (6). The true parameters a_{12} and b_{12} can be estimated as:

$$a_{12} = \widehat{a_{12}} \pm \frac{\Delta a_{12}}{2}, \tag{7}$$

$$b_{12} = \widehat{b_{12}} \pm \frac{\Delta b_{12}}{2}, \tag{8}$$

where

$$\widehat{a_{12}} = \frac{\overline{a_{12}} + \underline{a_{12}}}{2}, \tag{9}$$

$$\Delta a_{12} = \overline{a_{12}} - \underline{a_{12}}, \tag{10}$$

$$\widehat{b_{12}} = \frac{\overline{b_{12}} + \underline{b_{12}}}{2}, \tag{11}$$

$$\Delta b_{12} = \overline{b_{12}} - \underline{b_{12}}. \tag{12}$$

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 piggybacked on data messages. Since most MAC protocols in wireless networks employ an acknowledgment (ACK) scheme, the probes can be piggybacked 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, which is perhaps the most important constraint in wireless sensor networks).

B. Relaxing the immediate reply assumption on node 2

In Fig. 1 we assumed that node 2 replies immediately to node 1. The correctness of the presented approach is not affected in any way 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 cannot access the wireless channel, etc.).

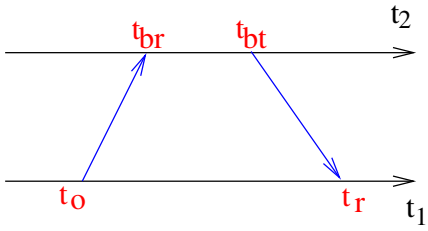


Fig. 3. A probe message from node 1 may be returned by node 2 and timestamped at both the send and receive points resulting in two data-points: (t_o, t_{br}, t_r) and (t_o, t_{bt}, t_r).

To counteract the possible loss in precision, node 2 can time-stamp the probe message both upon receipt (t_{br}) and when it resends it (t_{bt}) (as depicted in Fig. 3). In this case each of the triplets (t_o, t_{br}, t_r) and (t_o, t_{bt}, t_r) represent a data point satisfying (3) and (4). Thus we obtain two data-points using only one probe. They will be treated as two independent data-points and in this paper no further distinction will be made between the two methods of collecting data points.

C. Increasing the accuracy by considering the minimum delay

If no information about the delays encountered by the probe messages is available, nothing else can be done to increase the accuracy. However, if the minimum delay a probe encounters between the nodes is known, the data-points can be adjusted for a boost 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} the probe encounters between the moment t_o is stamped at node 1 and the moment t_b (t_{br}) is stamped at node 2. Also denote with δ_{21} the minimum delay between the moment t_b (t_{bt}) is stamped and the moment t_r is stamped. 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. both for the relative offset and the relative drift: $\Delta a_{12} = \Delta b_{12} = 0$).

III. 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 [4].

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 [4] 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 position estimates $[\underline{a_{12}}, \overline{a_{12}}]$ and $[\underline{b_{12}}, \overline{b_{12}}]$ are constrained only by the 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 we should be able to keep only four constraints (the ones which define the best bounds on the position estimates) 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 discarded (i.e. the four constraints which result in the best estimates are kept). The comparison operation to decide which four constraints to keep is very simple computationally (only 8 additions, 4 divisions and 4 comparisons). At any one time only the information for the best four constraints (8 timestamps = 4 constraints $\times$ 2 timestamps/constraint) 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 very well belong to two, three or four different data points.

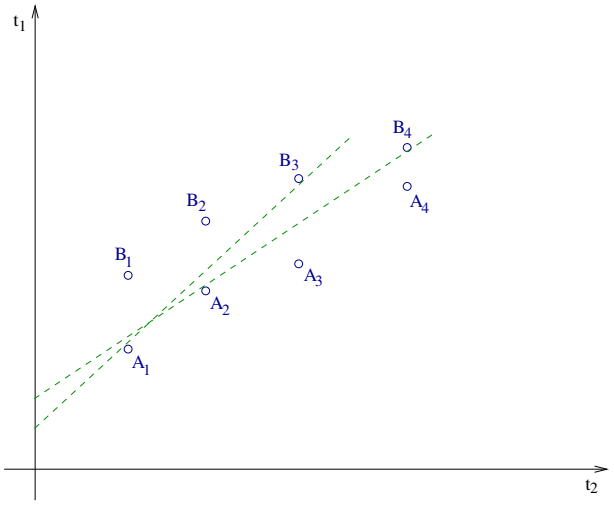


Fig. 4. Tiny-sync will miss the optimum solution in this situation, as it will discard the constraint A_2 upon the receipt of data-point $(A_3 - B_3)$.

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. 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 estimates improve, 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 tiny-sync while producing correct results, might miss the optimum result. We will compare the performance of tiny-sync with the optimal solution in the experimental Section V.

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. 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

$$m(A_i, A_j) > m(A_j, A_k) \tag{13}$$

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 (e.g. A_3) which satisfies*

$$m(A_i, A_j) \leq m(A_j, A_k) \tag{14}$$

for any integers $1 \leq i < j < k$ can be safely discarded as it will never be useful.

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. Since we only eliminate the constraints (conditions) 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 [4]).

Storing only four points like in tiny-sync does not produce the optimal solution. How many points do we actually need to store to produce the optimal solution? Theoretically a potentially large number. If the delay between node 1 and node 2 is monotonically increasing, (13) can hold for all the constraints A_j. In practice, the delays do not increase monotonically forever. Therefore, only a few constraints need to be stored to obtain the optimal result. In practice, our experiments showed that no more than 40 points have to be stored at any one time which is quite reasonable even for wireless sensor nodes.

IV. Synchronizing an Entire Network

In the previous Sections we presented two algorithms to synchronize two wireless sensor nodes. In this Section we will explore the possibility to extend the synchronization from two nodes to any number of nodes.

Consider the situation depicted in Fig. 5. Node s synchronizes with node u and node u synchronizes with node v. Therefore node s is able to determine the bounds

$$\underline{a_{su}} \leq \quad a_{su} \quad \leq \overline{a_{su}}, \tag{15}$$

$$\underline{b_{su}} \leq \quad b_{su} \quad \leq \overline{b_{su}}, \tag{16}$$

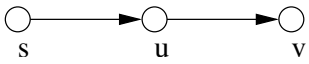


Fig. 5. Synchronization transitivity: if s is synchronized with u and u is synchronized with v, then s is synchronized with v.

and node u is able to determine the bounds

$$\underline{a_{uv}} \leq \quad a_{uv} \quad \leq \overline{a_{uv}}, \tag{17}$$

$$\underline{b_{uv}} \leq \quad b_{uv} \quad \leq \overline{b_{uv}}. \tag{18}$$

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

$$\underline{a_{sv}} \leq \quad a_{sv} \quad \leq \overline{a_{sv}}, \tag{19}$$

$$\underline{b_{sv}} \leq \quad b_{sv} \quad \leq \overline{b_{sv}}, \tag{20}$$

where

$$\underline{a_{sv}} = \underline{a_{su}a_{uv}}, \tag{21}$$

$$\overline{a_{sv}} = \overline{a_{su}a_{uv}}, \tag{22}$$

$$\underline{b_{sv}} = \min\left\{\underline{a_{su}b_{uv}} + \underline{b_{su}}, \overline{a_{su}}\underline{b_{uv}} + \underline{b_{su}}\right\}, \tag{23}$$

$$\overline{b_{sv}} = \max\left\{\overline{a_{su}}\overline{b_{uv}} + \overline{b_{su}}, \underline{a_{su}}\overline{b_{uv}} + \overline{b_{su}}\right\}. \tag{24}$$

This approach effectively synchronizes nodes s and v without exchanging any messages between them.

A wireless sensor network is logically organized as a hierarchy (see Fig. 6) with sensor nodes in the lowest layer, one (or more) root node(s) (also called sink(s)) and possibly several layers of intermediate nodes. The sensor nodes send the data at the intermediate node where the data is fused and further sent to the upper layers.

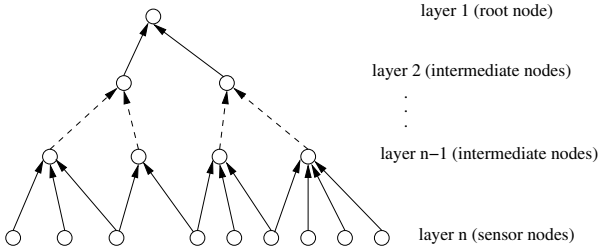


Fig. 6. Hierarchical wireless sensor network.

Since data is fused at the intermediate nodes, there is no point in synchronizing the entire network to one unique clock (e.g. the clock of the root node). Indeed, all the sensors reporting to the same intermediate node should synchronize with the fusion point and use the synchronized time to time-stamp all the data they send (eventually the lower and upper bounds on t_{2_0} and $\overline{t_{2_0}}$ are sent as time-stamps if so required by the application). In turn, the nodes in the intermediate layer i should synchronize to the nodes in layer $i-1$, and so on.

Synchronizing the nodes in layer i to the nodes in layer $i-1$ for all $i = 2, \ldots, n$ *is not* equivalent with

synchronizing all the nodes with the root node. If all nodes are synchronized with the root node, the precision of the synchronization degrades linearly with number of layers, the sensor nodes in layer n having the lowest precision. Instead, sensor nodes should synchronize only with the fusion nodes in the layer immediately above. Thus the precision can be kept relatively high (clearly better than if all nodes are synchronized with the root layer).

As suggested in Fig. 6, some sensors may report to more than one intermediate node. In this case the sensor nodes will synchronize with all intermediate nodes that receive data from it. How is it possible to synchronize with two intermediate nodes which are not themselves synchronized? This is indeed possible, because in the presented approach we do not modify the local clock. We just estimate the parameters a_{ij} and b_{ij} used to "correct" the local clock and estimate the time at the receiver of the data. To send data to two (or more) receivers, we'll just keep track of two (or more) sets of a_{ij} and b_{ij} and use each set for its corresponding receiver.

The logical organization of the wireless sensor network might not be mirrored by the physical layout of the nodes. For example, the sensors may send the data to the intermediate nodes through a multi-hop path which may go through other sensors, or even the root. In this case, the synchronization can be either done through transitivity (hop-by-hop) or the synchronization probes may be relayed on the same path that data will travel once the nodes are synchronized.

V. Experimental Results

To evaluate the performance of the proposed algorithms we implemented and tested them on two sets of data points. Since a wireless sensor network was not available, we used an 802.11b multi-hop ad-hoc network. It is expected that wireless sensor networks will exhibit similar delay patterns as the ones encountered in this ad-hoc network. To mimic the data traffic in the wireless sensor network, we used a background traffic in the ad-hoc network as well. For the first set of data points the two computers to be synchronized were neighbors (i.e. one hop away). For the second data set, the computers were five hops away. For both experiments a message probe was sent once a second for about 83 minutes thus resulting in almost 5000 data-point samples for each experiment. Each experiment was repeated several times to increase the confidence in the results. The statistics of the collected data-points are presented in Table I.

The same two sets of 5000 data points were used to compare the performance of the two algorithms. In practice far fewer messages have to be exchanged for precisions similar to the ones given by these 5000 data points, by taking the samples at the "right" times Also, in practice, only a limited number of data points (four for tiny-sync) have to be stored. In this section we worked with 5000 data points to observe the processing capabilities of the two algorithms.

TABLE I

Statistics of the data-points collected for the one hop and the five hops experiments.

	Min [ms]	Max [ms]	Avg [ms]	Std Dev [ms]
one hop	3.040	24.217	3.366	1.112
five hops	16.073	177.744	31.540	12.874

TABLE II

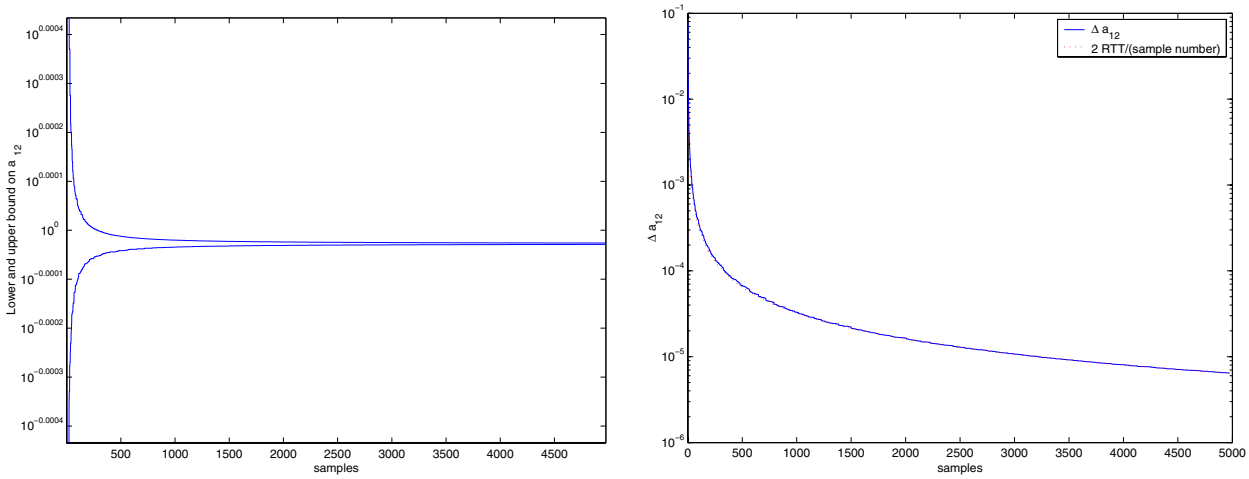
Upper bounds of the relative error between mini-sync and tiny-sync

	Relative errors for			
	a_{12}	$\overline{a_{12}}$	b_{12}	$\overline{b_{12}}$
one hop	0.14%	0.19%	0.19%	0.14%
five hops	1.8%	1.7%	1.7%	1.8%

Fig. 7 a) and 8 a) depict the evolution of the bounds on a_{12} and b_{12} respectively. Fig. 7 b) and 8 b) depict the improvement in precision of the estimate Δa_{12} and Δb_{12} (see (7) and (8)) respectively. The values correspond to the five hops data points processed with mini-sync. The corresponding figures for one hop are similar. As expected, the bounds converge. It can be shown that precision of the drift is expected to be bounded by $2RTT/(sample\_number)$ where RTT is the round trip time of the samples, and $sample\_number$ is the number of samples collected. Also the precision of the offset is expected to be on the order of RTT. The justification for both expected bounds is omitted due to the lack of space. The expectations match closely the experimental results in Fig. 7 (b) and Fig. 8 (b). The precision of the offset estimate increases significantly during the first samples, and then improves only marginally. In contrast the precision of the drift continues to improve significantly for as long as data samples are collected.

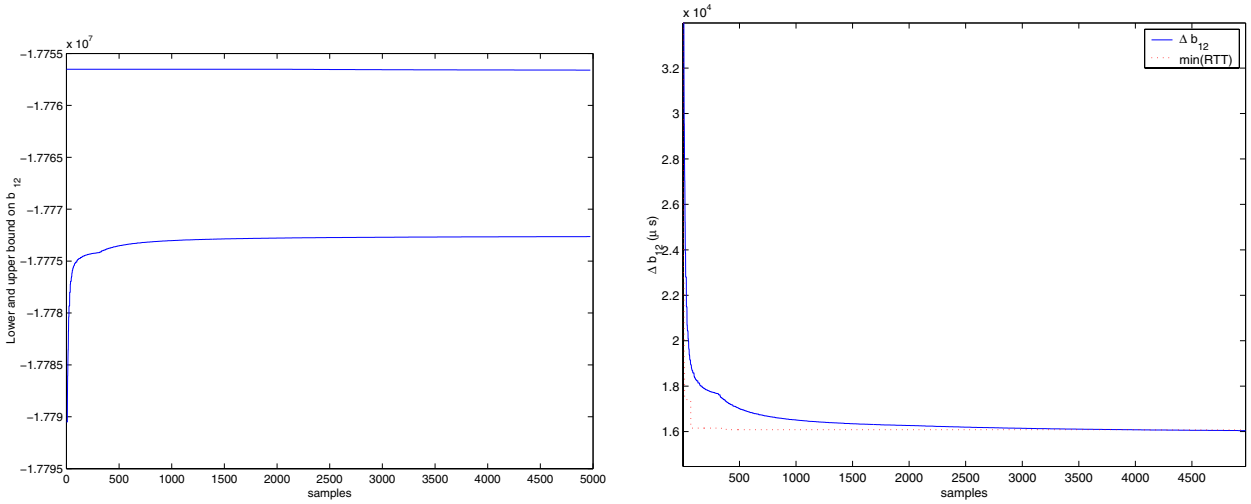
In theory tiny-sync, while simple, is sub-optimal. To evaluate the loss in precision for tiny-sync we computed the upper bound of the relative error between mini-sync (the optimal solution) and tiny-sync after each new sample was processed. The results are presented in Table II. It can be seen that the difference is practically insignificant - smaller than 2% in all cases (and actually smaller than 0.1% after the first few samples have been processed). The direct implication is that in practice usually one does not need the precision provided by mini-sync. The results from tiny-sync are very close to the ones from mini-sync with far less complexity.

We can improve the results if we take into account the minimum delays that 802.11b introduces for each hop. We used 256 bytes UDP packets in both directions (to mimic the piggybacking of the time-stamps



(a) (b)

Fig. 7. a) Evolution of the bounds on a_{12} (a) and of Δa_{12} (b) as more samples are collected



(a) (b)

Fig. 8. a) Evolution of the bounds on b_{12} (a) and of Δb_{12} (b) as more samples are collected

on sensed data and control information) and we used the CSMA/CA high rate (11 Mbps) mode of 802.11b. A careful examination of all delays involved (DIFS, preamble, MAC, LLC/SNAP header, IP and UDP headers) results in a minimum transmission delay for each probe of 1089 μs. For the five hop case, neglecting the processing in each intermediate node, we consider the minimum one way delay equal to five minimum transmission times.

To evaluate the usefulness in adjusting the results for the minimum delay, in Table III the results of tiny-sync are shown after processing both streams of data samples. The results for mini-sync are similar.

The improvement obtained by eliminating known delays is significant (up to five times better), and it can be further increased especially for wireless sensor network nodes where many of the uncertainties present in the ad-hoc network used as testbed can be eliminated. In the one hop case we were able, in a little over an hour, to bound the offset by $\pm 945 \mu s$ and the drift by $\pm 2.7\ 10^{-7}$ corresponding to a drift of 23.3 ms in a day.

TABLE III
RESULTS FOR TINY-SYNC WITH AND WITHOUT DATA PRE-PROCESSING

	Raw data		Pre-processed data	
	$\frac{\Delta a_{12}}{2}$	$\frac{\Delta b_{12}}{2}(ms)$	$\frac{\Delta a_{12}}{2}$	$\frac{\Delta b_{12}}{2}(ms)$
one hop	7.133e-07	2.0941	2.768e-07	0.9457
five hops	5.013e-06	17.08	1.167e-06	3.239

VI. Conclusion

A light-weight synchronization algorithm is presented. The proposed algorithm is able to produce tight, deterministic synchronization with only few message exchanges. While the algorithm is suitable for any type of network, it is especially useful in wireless sensor networks which are typically extremely constrained on the available computational power and bandwidth and have some of the most exotic needs for high precision synchronization. The performs of the presented algorithms is verified with an experimental testbed. The experimental results match closely the theoretical expectations. A method to extend the synchronization to the entire sensor network, as needed for data fusion is also presented. Two varieties of the algorithm are presented and their performance is compared using real delay traces collected from a wireless ad-hoc network. The experimental results show that the simplest of the two algorithms, called tiny-sync, produces results very close to the optimum (within 0.1%) and thus is preferable.

References

[1] G. Asada, M. Dong, T. Lin, F. Newberg, G. Pottie, W. Kaiser, and H.O.Marcy, "Wireless integrated network sensors: low power systems on a chip," in *Proc. of the 24th European Solid-State Circuits Conference*, The Hague, Netherlands, Sept. 1998.

[2] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, *Global Positioning System: Theory and Practice*, 4th ed. Springer-Verlag, 1997.

[3] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[4] M. Lemmon, J. Ganguly, and L. Xia, "Model-based clock synchronization in networks with drifting clocks," in *Proc. of the 2000 Pacific Rim International Symposium on Dependable Computing*, Los Angeles, CA, Dec. 2000, pp. 177–185.

[5] R. Ostrovsky and B. Patt-Shamir, "Optimal and efficient clock synchronization under drifting clocks," in *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing(PODC99)*, 1999.

[6] J. Lundelius and N. Lyunch, "A new fault-tolerant algorithm for clock synchronization," *Information and Computation*, vol. 77, no. 1, pp. 1–36, 1988.

[7] L. Lamport and P. J. Melliar-Smith, "Synchronizing clocks in the presence of faults," *Journal of the ACM*, vol. 32, no. 1, pp. 52–78, 1985.

[8] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146–158, 1989.

[9] R. Gussell and S. Zatti, "The accuracy of clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD," *IEEE Transactions on Software Engineering*, vol. 15, pp. 847–853, 1989.

[10] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J-ACM*, vol. 34, no. 3, pp. 625–645, 1987.

[11] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Communications*, vol. 39, no. 10, pp. 1482–1493, Oct. 1991.

[12] ——, "Improved algorithms for synchronizing computer network clocks," in *Proc. of ACM Conference on Communication Architectures (ACM SIGCOMM'94)*, London, UK, Aug. 1994.

[13] P. Ashton, "Algorithms for off-line clock synchronization," in *Technical Report TR COSC 12/95, Department of Computer Science*, University of Canterbury, Dec. 1995.

[14] A. Duda, G. Harrus, Y. Haddad, and G. Bernard, "Estimating global time in distributed systems," in *Proc. of the 7th IEEE International Conference on Distributed Computing Systems (ICDCS'87)*, Berlin, Germany, Sept. 1987.

[15] J. Elson and D. Estrin, "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, Apr. 2001.

[16] K. Römer, "Time synchronization in ad hoc networks," in *Proc. of ACM Mobihoc*, Long Beach, CA, 2001.

[17] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *UCLA Technical Report 020008*, Feb. 2002. [Online]. Available: citeseer.nj.nec.com/elson02finegrained.html

[18] D. Dolev, J. Halpern, B. Simmons, and H. Strong, "Dynamic fault-tolerant clock synchronization," *Journal of the ACM*, vol. 42, no. 1, pp. 1–36, 1988.

Appendix

Proof of Theorem 1

We can provide a strictly mathematical proof for Theorem 1. However, there is little intuition behind the mathematical proof. Instead, we will provide a graphic argument.

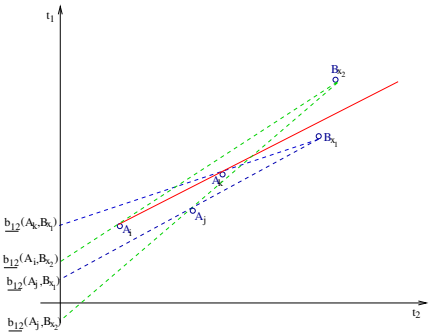


Fig. 9. A_j can be safely discarded as A_i or A_k will result in better estimates in all cases.

Consider the situation depicted in Fig. 9. The points A_i, A_j and A_k satisfy condition (14). Consider a point 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 any of the constraints A_i, A_j or A_k may determine a lower bound on b_{12} (or alternatively an upper bound on a_{12}). Denote with $\underline{b_{12}}(A_y, B_x)$ the lower bound on b_{12} determined by the constraints A_y and B_x. We can distinguish two cases:

1. If B_x is *below* the line determined by A_i and A_k (position B_{x_1} in Fig. 9). In this case $\underline{b_{12}}(A_j, B_{x_1}) < \underline{b_{12}}(A_k, B_{x_1})$ and the constraint A_j can be safely discarded.
2. If B_x is *above* the line determined by A_i and A_k (position B_{x_1} in Fig. 9). In this case $\underline{b_{12}}(A_j, B_{x_2}) < \underline{b_{12}}(A_i, B_{x_2})$ and the constraint A_j can be safely discarded as well.

Thus in both cases, the constraint A_j can be discarded as it will never be useful (i.e. it will never constrain a_{12} or b_{12} more than than A_i and A_k do).