

Analysis and Performance Evaluation of a Time Synchronization Protocol for Wireless Sensor Networks

Suyoung Yoon and Mihail L. Sichitiu

Department of Electrical and Computer Engineering

North Carolina State University

Raleigh, NC 27695-7911

{syoon2,mlsichit}@ncsu.edu

Abstract—

Time synchronization is a fundamental middleware service for any distributed system. Wireless sensor networks introduce new requirements to this age-old problem regarding its scalability, cost and precision. In this paper we analyze theoretically and evaluate the performance of a very simple time synchronization algorithm. The performance of the algorithm is deterministic and predictable. To evaluate its performance we implemented the algorithm on Mica2 Berkeley nodes. The results show that the algorithm achieves very good precision with fewer resources than existing algorithms.

I. INTRODUCTION

Recent technological advances in low power radios, sensors and microcontrollers enabled 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) [1]–[3] have the potential to truly revolutionize the way we monitor and control our environment.

Many (if not most) WSN applications either benefit or require time synchronization. Sensed data is typically forwarded over multiple hops to one or several base stations and encounters delays ranging from a few tens of milliseconds to several minutes [4]. Many power saving algorithms trade power savings for increased delays. Therefore, the time sensed data reached the base station is often a poor approximation of the time the data was sensed. If the entire sensor network is synchronized (to the base station), timestamps can accompany sensed data. Many WSN protocols (e.g., power saving sleep schedules [5], TDMA schedules for maximizing the networks' capacity, security mechanisms for preventing replay attacks, etc.) also rely on time synchronization.

In this paper we focus on *pair-wise* time synchronization, i.e., synchronizing just two wireless sensor nodes within wireless range of each other. It was shown [6], [7] that the entire network can be globally synchronized using pair-wise synchronization by forming a synchronization tree. In WSNs, such trees are normally formed for routing the data to and from the base stations. If for some reasons base stations are unavailable, a leader election algorithm can be employed [7].

In our previous work [6] we introduced tiny-sync (a brief overview is presented in Section III), a simple time-synchronization algorithm and compared its performance with an optimal time-synchronization algorithm. The comparison showed that tiny-sync performs very close to optimal (within 0.1%) at significantly less computational and storage costs. In this paper we analyze tiny-sync and evaluate its performance with a realistic testbed.

II. 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) [8], [9] has been widely used in the Internet for decades. However, traditional synchronization schemes 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 [10]. 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** is of primary concern in WSNs as nodes have typically limited batteries, computational and storage resources. Most of the protocols designed for wired environments exchange many messages (for statistical processing). Furthermore, they also need to store those messages (to allow for their processing).

Recently, a significant amount of research on time synchronization for wireless sensor has been published [11], [12]. An interesting approach called *post facto synchronization* was proposed by Elson and Estrin [13]. 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 (timestamp 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.

In the reference broadcast synchronization (RBS) protocol [14], the network topology assumes a beacon node capable of broadcasting over the entire network. The beacon node periodically sends reference packets and all other nodes record the packet arriving time. The nodes then exchange their recorded timestamps and estimate a global time using linear regression. The interesting feature of RBS is that it records the timestamp only at the receivers. Thus, all timing uncertainties on the transmitter's side are eliminated. The main disadvantage of RBS protocol is that all network nodes should be in the transmission range of the beacon node and then be to communicate with each other.

Römer [15] presented a synchronization protocol for ad hoc networks. The authors assume clocks with known maximum clock drift. The basic idea of the algorithm is to compute timestamps using unsynchronized local clocks. When a local timestamp 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”. The protocol was enhanced in [16] resulting in increased accuracy and reduced resources.

A significant challenge when implementing time synchronization protocols is minimizing timestamping uncertainties. RBS reduces these uncertainties by timestamping only at the receivers. The time synchronization protocol for sensor network (TPSN) [17] reduces the uncertainties by using timestamps at the medium access control (MAC) layer. This eliminates the (large) uncertainties introduced by the MAC layer (retransmissions, backoffs, yields, etc). TPSN takes advantage of the availability of the MAC layer code in TinyOS [18]. TPSN offers a two-fold increase in precision in comparison to RBS.

The Flooding Time Synchronization Protocol (FTSP) [7] was designed for a sniper localization application requiring very high precision [10]. FTSP achieves the required accuracy by utilizing a customized MAC layer timestamping and by using calibration to eliminate unknown delays. FTSP is robust to network failures, as FTSP uses flooding both for pairwise and for global synchronization. Linear regression from multiple timestamps are 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, the FTSP's precision is impressive (less than $2\mu s$).

Lightweight Time Synchronization (LTS) [19] was proposed for applications where the required time accuracy is relatively low. The pairwise 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 result shows that the accuracy of LTS is about 0.5 seconds.

Li and Rus [20] 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 [21] 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 dramatically reducing the number of message exchanges). Similar to RBS, the beacon has to be capable of covering 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.

In our previous work [6], we introduced tiny-sync, a simple and accurate time synchronization protocol for WSNs. In this paper, we provide analytical results for tiny-sync including the precision bound and practical synchronization strategies. We also evaluate its performance using a Mica2 [22] mote implementation. Tiny-sync features the following combination of desirable properties not available in any other existing time synchronization protocols:

- Tiny-sync is inexpensive in terms of message exchange, computational and storage requirements.
 - Only four timestamps are stored for pairwise synchronization.
 - Only minimal computations are required for every data-point.

- Tiny-sync does not require access to the MAC layer, while, however, benefiting from such access. Thus it can be truly hardware independent. With limited MAC layer access it performs slightly better than any of the existing time synchronization protocols.
- Tiny-sync shows less dependency on the synchronization period by comparison to other existing protocols.

III. TINY-SYNC OVERVIEW

In this section, we provide a brief overview of tiny-sync [6]. Assume two wireless sensor nodes (1 and 2) with monotonically increasing local clocks $t_1(t)$ and $t_2(t)$. Assume for the moment that the clocks are perfectly linear:

$$t_k(t) = a_k t + b_k, \quad k \in \{1, 2\}, \quad (1)$$

where a_k and b_k are the drift and the offset of node k 's clock and t is the universal coordinated time (UTC). This is a reasonable assumption for a limited time if the local clocks are driven by quartz oscillators. We will analyze the consequences of the failure of this assumption in Section IV. The immediate consequence of the assumption (1) is that t_1 and t_2 are linearly related:

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

where a_{12} and b_{12} represent the *relative drift* and the *relative offset* between the two clocks respectively.

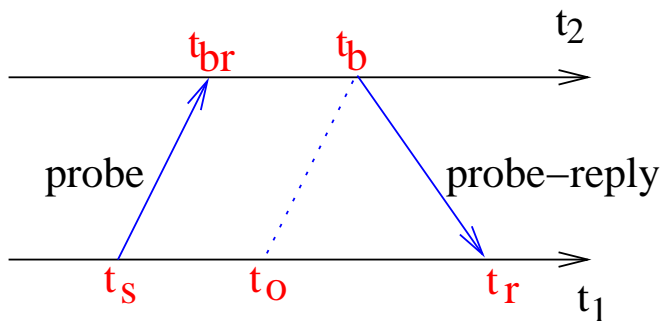


Fig. 1. A probe message from node 1 is returned by node 2 and timestamped at both the send and receive points resulting in four timestamps: (t_o, t_{br}, t_b, t_r) .

Assume that node 1 has to estimate the local time at node 2. Node 1 will send a timestamped (at time $t_s = t_1(\text{probe-transmission})$) probe message to node 2. Node 2 will respond with a probe reply message including the

timestamps at the reception of the probe message and the transmission of the probe reply (using the local time at node 2: $t_{br} = t_2(\text{probe-reception})$ and $t_b = t_2(\text{probe-reply-transmission})$ respectively). Finally, node 1 provides a final timestamp $t_r = t_1(\text{probe-reply-reception})$ at the moment of receiving the probe reply. Figure 1 depicts such a timestamp exchange between nodes 1 and 2.

We call the data collected from one probe exchange (the four time stamps) a *data point*. Let us denote $t_o = t_s + a_{12}(t_b - t_{br})$; t_o is the latest time at node 1 that is known to be before t_b . Each data point imposes two constraints on the values of the two unknowns a_{12} and b_{12} in (2):

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

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

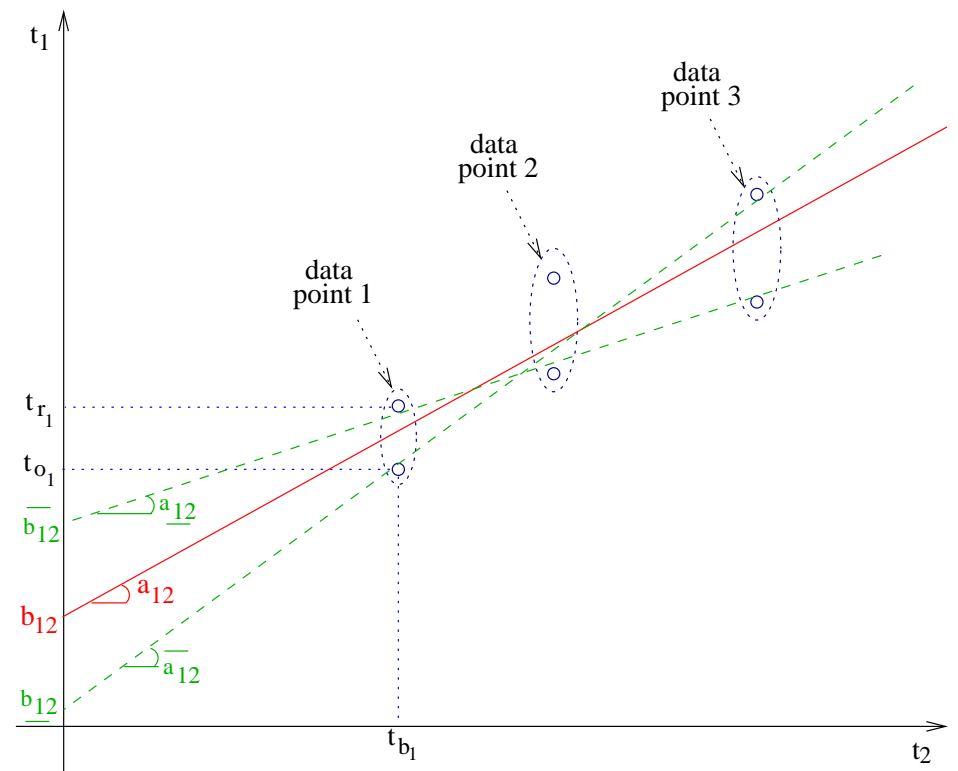


Fig. 2. The linear dependence (2) and the constraints imposed on a_{12} and b_{12} by three data-points.

The two constraints (3) and (4) are essentially causality constraints (i.e., one timestamp happened before the next one). The constraints have an intuitive graphical representation as the set for the feasible values of a_{12} and b_{12} has to be positioned between two constraints (t_b, t_o) and (t_b, t_r) (see Fig. 2 [6]). It is clear that the more data points are available, the more constrained are the values of a_{12} and b_{12} .

Optimally, all constraints should be simultaneously considered. This was shown to result into two linear programming problem with twice as many inequalities as data points [23]. It was shown [6] that considering only the best two data points instead of all of them results only in a negligible reduction in precision (less than 0.1%). Hence, tiny-sync only stores two data points at any one time. When a new data point is received, the three data points (two stored and one received) are compared and only the two data points that constrain the values of a_{12} and b_{12} the most are retained.

The exact values of a_{12} and b_{12} cannot be exactly determined using this approach (or any other approach) as long as some components of the delays between the causal timestamps remain unknown (e.g., propagation time, medium access, interrupt service time, etc.). However, a_{12} and b_{12} can be bounded:

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

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

The true parameters a_{12} and b_{12} can be estimated as:

$$a_{12} = \widehat{a_{12}} \pm \frac{\Delta a_{12}}{2}, \quad (7)$$

$$b_{12} = \widehat{b_{12}} \pm \frac{\Delta b_{12}}{2}, \quad (8)$$

where

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

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

$$\widehat{b_{12}} = \frac{\overline{b_{12}} + \underline{b_{12}}}{2}, \quad (11)$$

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

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.

As shown in [6], [7], pairwise synchronization can be used to achieve global synchronization of the entire network (with respect to one node, usually a base station). The global synchronization can proceed in parallel as nodes compute *relative* drifts and offsets; given the drift and offset calculation of a node u with respect to a global reference and the relative drift of the node v to node u , it is trivial to compute the drift and offset of v with respect to the global reference [6]. Thus, it takes the same amount of time to synchronize a network as it takes to synchronize just a pair of nodes.

Tiny-sync is robust to wireless transmission errors: if a probe reply is not received, node 1 can send a new probe (or just use the existing data points - for a slight decrease in accuracy). In other words, tiny-sync does not rely on receiving all packets correctly.

IV. ANALYSIS OF TINY-SYNC

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

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

$$t_{r_i} - t_{o_i} = RTT \quad \forall i \geq 1. \quad (13)$$

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 constrain our variables the most, 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);
- as shown in Section VI, the theoretical analysis matches the experimental results exceedingly well (to the point that the prediction and the results are indistinguishable on the graphs).

With assumption (13), the data points that define the best approximation are always the first and the last data-points collected. All the other data-points can be safely discarded. As a side effect, tiny-sync's performance with

this assumption is identical to the performance of the optimal algorithm (that considers all the constraints). Denote with $(t_{o_1}, t_{b_1}, t_{r_1})$ and $(t_{o_2}, t_{b_2}, t_{r_2})$ the first and the last collected data points respectively.

Figure 3 presents the graphical interpretation of tiny-sync variables relevant to the following analysis.

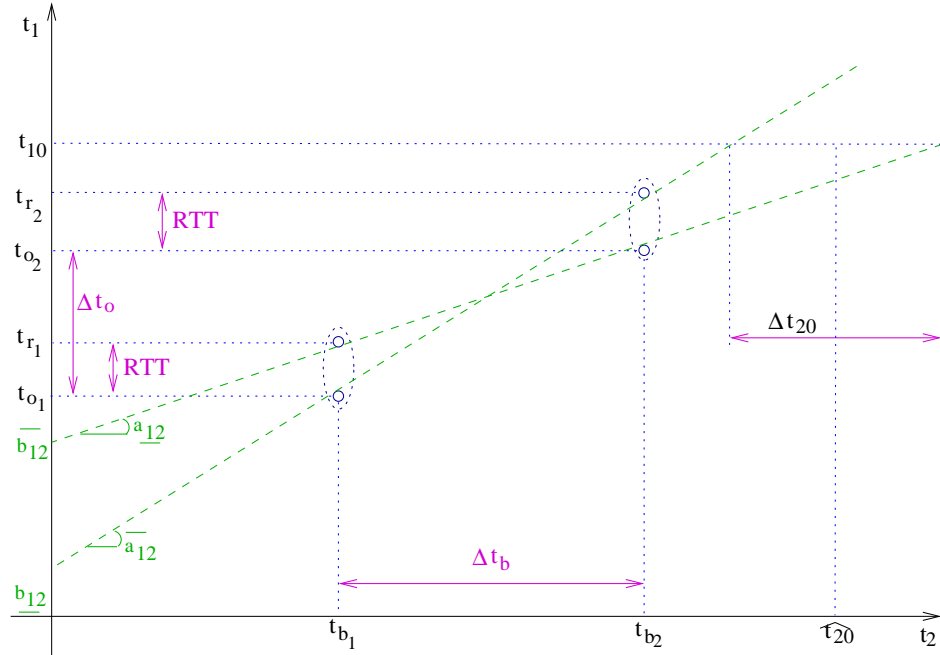


Fig. 3. Setup for the simplified analysis of the algorithm.

Using the notations in Fig. 3 and solving the linear equations for the unknowns \underline{a}_{12} , \overline{a}_{12} , \underline{b}_{12} and \overline{b}_{12} we obtain:

$$\underline{a}_{12} = \frac{\Delta t_o - RTT}{\Delta t_b}, \quad (14)$$

$$\overline{a}_{12} = \frac{\Delta t_o + RTT}{\Delta t_b}, \quad (15)$$

$$\underline{b}_{12} = t_{o_1} - t_{b_1} \underline{a}_{12}, \quad (16)$$

$$\overline{b}_{12} = t_{o_1} + RTT - t_{b_1} \underline{a}_{12}. \quad (17)$$

Using (10), (12) we obtain the following bounds on precision of the estimates

of a_{12} and b_{12} :

$$\Delta a_{12} = \frac{2RTT}{\Delta t_b}, \quad (18)$$

$$\Delta b_{12} = RTT + t_{b_1} \Delta a_{12}. \quad (19)$$

Equations (18) and (19) capture the essential behavior of tiny-sync:

Equation (18) shows that the uncertainty bound on the drift is expected to decrease continuously as the distance between the first and the last collected data-points increases:

$$\lim_{\Delta t_b \rightarrow \infty} \Delta a_{12} = 0. \quad (20)$$

Figure 4 depicts the expected evolution of the uncertainty bound on the relative clock drifts: Δa_{12} remains constant between data-points, and improves upon the receipt of a new data point.

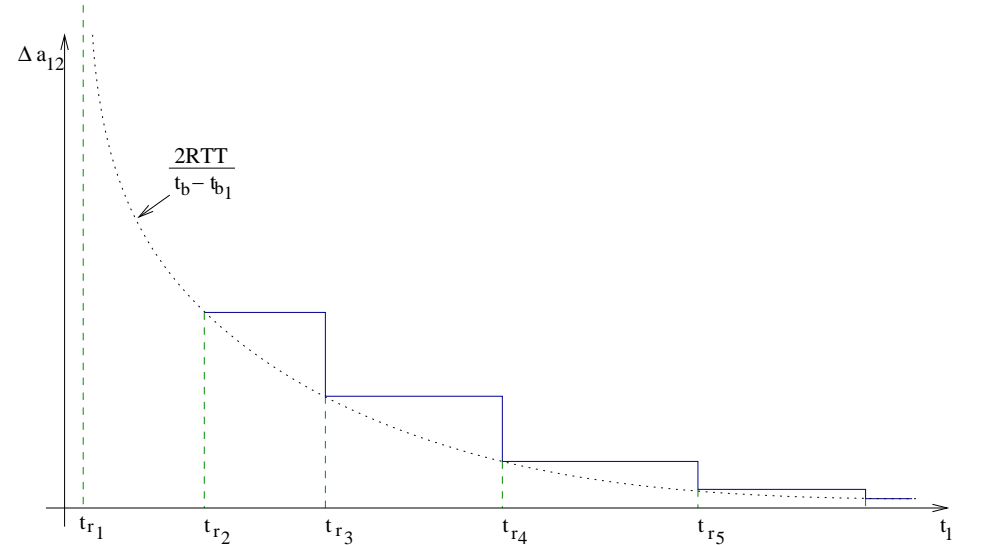


Fig. 4. Evolution of the uncertainty bound on the relative clock drifts Δa_{12} as new data-points are received.

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

$$\lim_{\Delta t_b \rightarrow \infty} \Delta b_{12} = RTT. \quad (21)$$

The second, implication of (19) 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:

$$t_{b_1} = 0. \quad (22)$$

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

A. 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. Tiny-sync is constructed on the assumption of linear drifts. Thus, it is important to *detect* when assumption (1) no longer holds.

Fortunately, 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 (18)). 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).

Thus, practically speaking, during the algorithm, after each probe, the computed Δa_{12} should be 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

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

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

V. EXPERIMENTAL SETUP

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

Tiny-sync was implemented on the Telos platform [24], the latest generation in the family of motes from UC Berkeley, using TinyOS [18] version 1.1.12 (featuring reduced delay uncertainties by allowing MAC layer timestamping and byte alignment correction). In the experiment, the external 32KHz crystal clock was used to obtain current timestamps. We also ran tiny-sync on the Mica2 platform [22] where the clock source was 7.3MHz internal clock. In the following sections, we mainly explain the results of Telos platform because the external clock is more stable than internal one.

A. Sensor nodes configuration

As we explained in Section I, tiny-sync only considers pairwise synchronization as the basic building block for global network synchronization. The experimental setup is, hence, designed to test the performance of the pairwise synchronization.

Three Mica2 motes were used to implement the tiny-sync algorithm and to measure the synchronization errors. Figure 5 depicts the experimental setup: nodes 1 and 2 run the tiny-sync algorithm. In our experiment, node 1 estimates node 2's clock. The third mote (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.

B. Tiny-sync implementation

Figure 6 shows the moments of recording the four timestamps in Section III. We used the hooks provided by TinyOS to timestamp the packet immediately after the frame sync byte was transmitted, and, respectively received. Thus all the non-deterministic delays (RTT in Section IV) are due to the transceiver. Our measurements showed that the delay introduced by the transceivers (including the encoding and decoding) was about $550 \mu s$ ($1100 \mu s$ for the round trip time); the exact value is *not* available on the Chipcon CC1000 data-sheet [25] of the transceiver. The round trip time variation is about $38 \mu s$. These values do not match the values published by other research teams [7]. We attribute

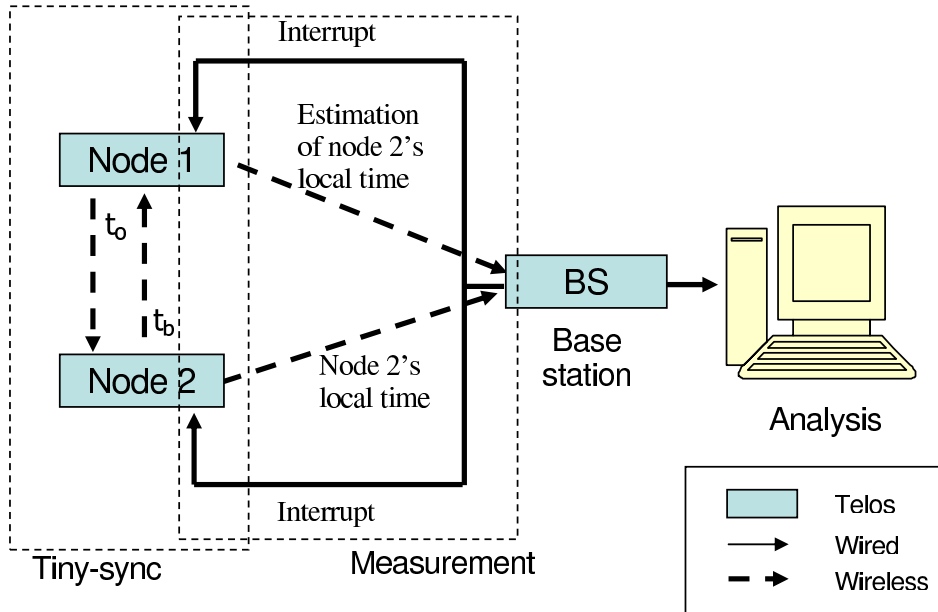


Fig. 5. Experimental setup.

this discrepancy to different versions of the hardware and/or the software used by the two teams.

C. Measuring synchronization errors

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 sources of errors when measuring the synchronization error:

- the synchronization algorithm, and
- the measurement procedure itself.

As mentioned in Section V in our 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 7 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 jumps to the corresponding

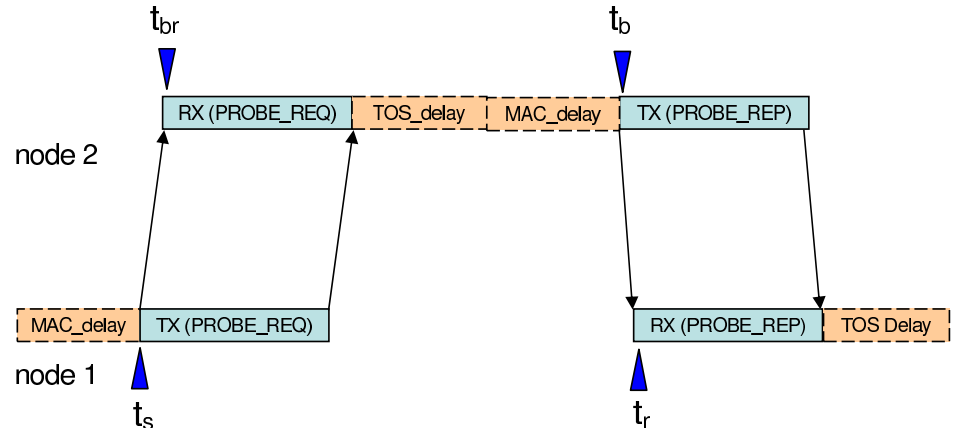


Fig. 6. Timestamping in tiny-sync.

interrupt service routine (ISR). In our case the IRS reads the current clock. Unfortunately, if the processor was in the middle of a non-interruptible operation (i.e., the interrupts were suspended) or there are other pending interrupts, the measurement interrupt will be delayed an unknown time (in our experiments we found it being up to one millisecond).

There are several ways to define the measurement error. We define the measurement error, $E_i(M_j)$ of the measurement j at node i as

$$E_i(M_j) = x_{i,j} - x_{i,j-1} - \rho_i, \quad (24)$$

where $x_{i,j}$ are the timestamps of node i at measurement j and ρ_i is the most frequent value of the inter-measurement times at node i . That is, the measurement error is the difference between current inter-measurement time and the most frequent value of all inter-measurement times. 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 $5\mu s$ (i.e., we discarded all measurements with an error higher than $5\mu s$). A percentage of 36.7% of the measurements have their errors smaller than the threshold value.

VI. EXPERIMENTAL RESULTS

In this section, we present the results of the experiments we performed to evaluate the performance of tiny-sync and compare them with other protocols.

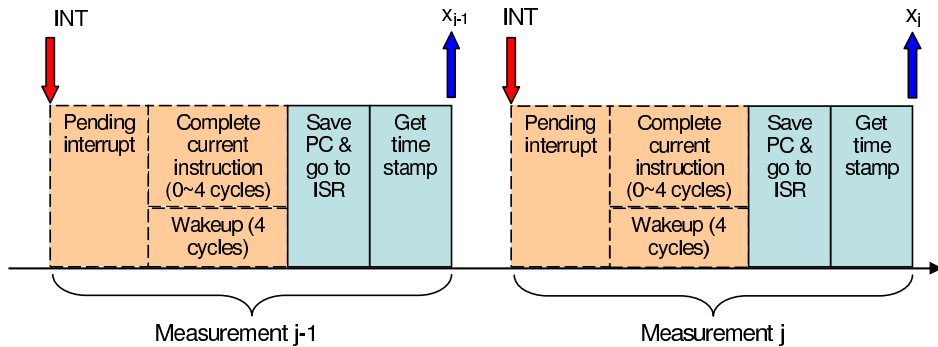


Fig. 7. Sources of delay in the measurement process error.

Using the setup described in Section V we ran the first experiment for 14.7 hours.

A. Time synchronization error of tiny-sync

Figure 8 shows the bound on the relative drift of the two clocks Δa_{12} and $\frac{2RTT}{\Delta t_b}$ in (18). It is clear that the relation (18) holds exceedingly well, implying that the assumption (1) holds within the bounds of our accuracy for this first experiment. In Section VI-C we present the results of another experiment where this assumption is broken on purpose.

Figure 9 shows the synchronization error of tiny-sync as a function of time. As discussed, the relative error is computed as the difference between the local time of node 2 and the estimated time of node 2 as computed by node 1. The average synchronization error achieved by tiny-sync is $1.3868 \mu s$. The algorithm converges quickly to the nominal precision (it achieves the average precision with the first few probes). Considering that one can synchronize the entire network as fast as synchronizing a pair of nodes (Section III), only a few seconds are necessary to synchronize an entire network (it is likely that, for large networks, the transmission of the relative offsets and clock drifts is the actual bottleneck rather than the synchronization algorithm).

Figure 10 shows the distribution of the synchronization errors. The graph indicates that almost 98.74% of time synchronization errors are smaller than $10 \mu s$.

B. Comparison with other approaches

In this section, we compare the performance of tiny-sync with other existing time synchronization approaches for WSNs.

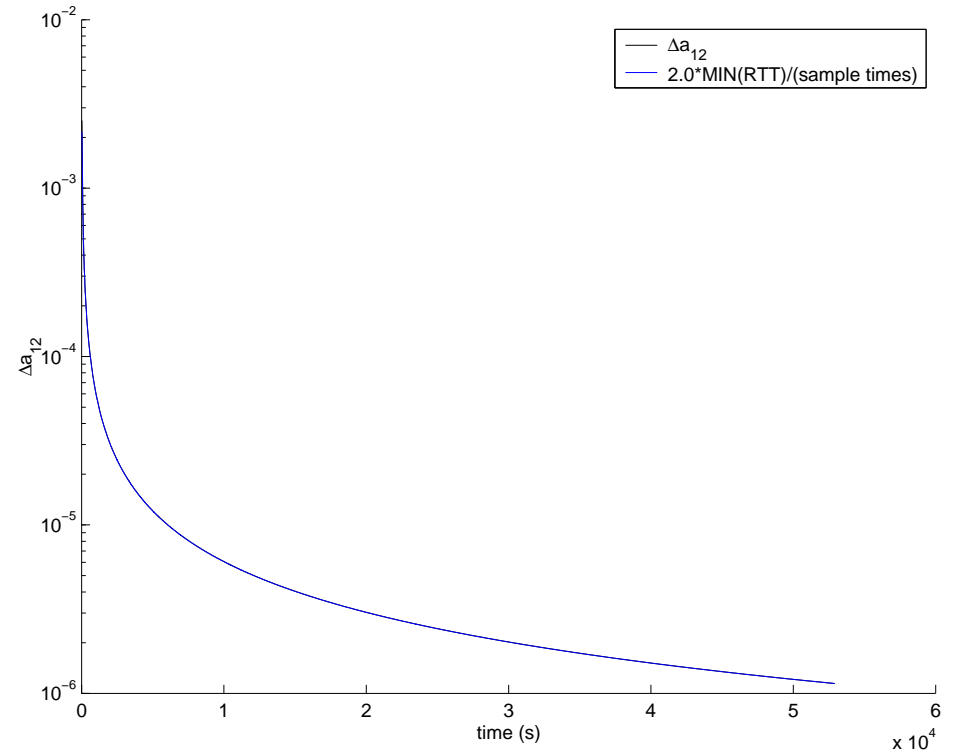


Fig. 8. The evolution of the bound on the relative drift Δa_{12} and the predicted evolution (18) for the first experiment.

According to published data [7], [17], the pair-wise accuracies of RBS, TPSN and FTSP are $29.1 \mu s$, $16.9 \mu s$ and $1.48 \mu s$ respectively. The accuracy of tiny-sync is very 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).

Since tiny-sync and TPSN use the same data collection method, we processed *the same* measurements from the first experiment using tiny-sync and TPSN. With a synchronization period $T = 4s$ the average error of TPSN was surprisingly good: $1.72 \mu s$. This is in direct contrast with the published results ($16.9 \mu s$ [17]). One explanation for this discrepancy may be attributed to the more careful timestamping enabled by upgrades in TinyOS. Another explanation may be related to measurements errors: we filtered out many of

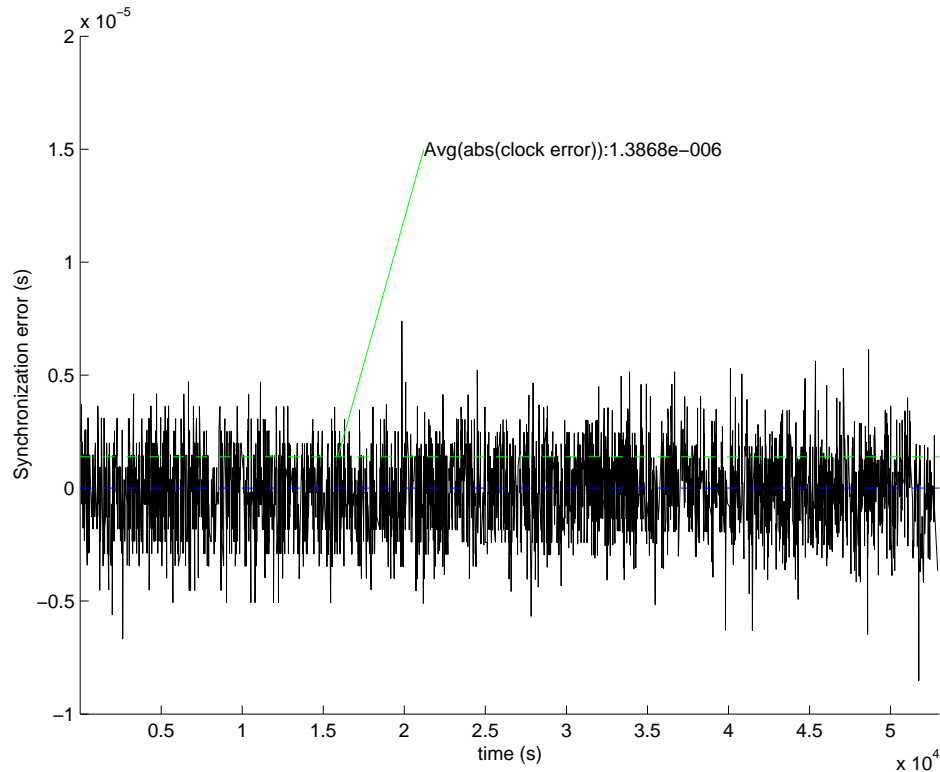


Fig. 9. The synchronization error for the first experiment.

the erroneous measurements that may appear to reduce the accuracy of the algorithm. Figure 11 shows the evolution of the synchronization error of TPSN.

However, the accuracy of TPSN quickly degrades as the synchronization period increases as shown in Fig. 12. The accuracy of both approaches decreases as the synchronization period increases, however tiny-sync's accuracy is significantly better than TPSN's at higher synchronization periods.

C. Robustness of tiny-sync

In this section, we present the results of an experiment that tests the robustness of tiny-sync with respect to large drifts in the clock. The duration of this experiment was 88.8 hours. For this experiment, we placed node 2 in a cooler filled with ice; we then closed and opened the lid several times. The change in temperature (when node 2 was introduced in the cooler, but

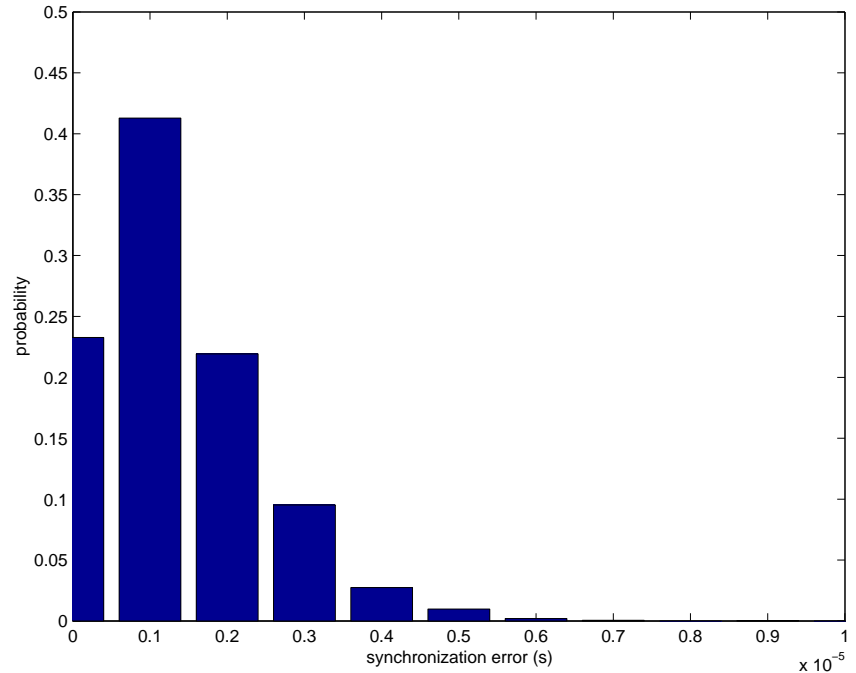


Fig. 10. Distribution of the synchronization errors.

also when we opened and closed the lid), as expected, negatively affected the relative drift of the clock. We used the procedure detailed in Section IV-A to detect the change in the drift of the clocks and restart the algorithm. Figure 13 shows the moments when the algorithm detected the change in the relative drift of the clocks and restarted.

Figure 14 shows the synchronization error for the second experiment. As expected the errors are larger than in the first experiment (when both nodes were next to one another in an air conditioned room). The average synchronization error is $4.8 \mu s$. The areas with significant errors correspond to changes in the relative clock drift; those changes were detected and the algorithm restarted.

VII. CONCLUSION

In this paper, we analyzed and evaluated the performance of a simple time synchronization algorithm suitable for wireless sensor networks. The algorithm performs pairwise synchronization and can be used as the basic building block

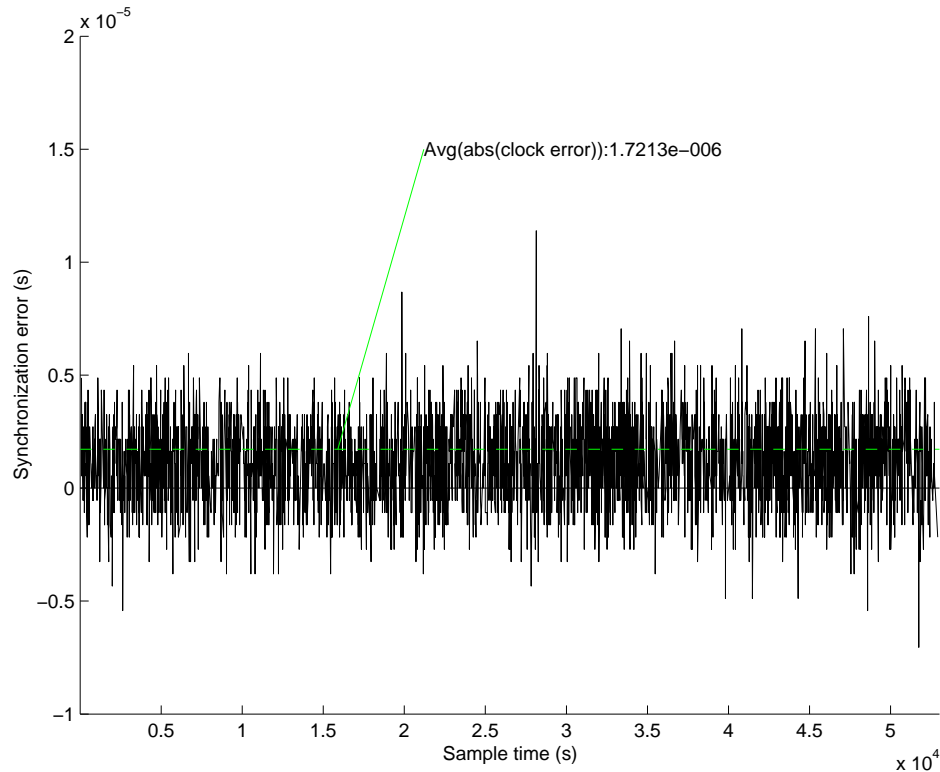


Fig. 11. Synchronization error of TPSN.

for synchronizing an entire network. The main advantage of the proposed approach is its simplicity and its parsimonious resources requirements. Other advantages include its 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

- [1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A survey on sensor networks," *IEEE Communication Magazine*, vol. 40, no. 8, pp. 102–116, Aug. 2002.
- [2] —, "Wireless sensor networks: A survey," *IEEE Computer*, vol. 38, no. 4, pp. 393–422, Mar. 2002.
- [3] G. Asada, T. Dong, F. Lin, G. Pottie, W. Kaiser, and H. Marcy, "Wireless integrated network sensors: Low power systems on a chip," in *Proc. of the 24th European*

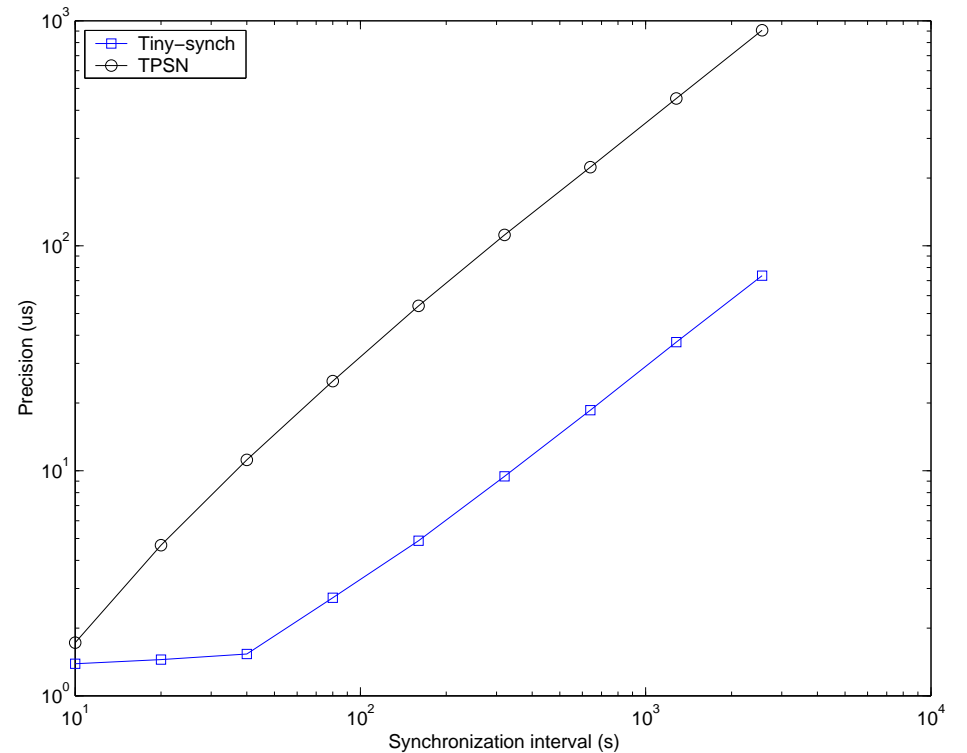


Fig. 12. Variation of the accuracy of tiny-sync and TPSN with the synchronization period.

- Solid-State Circuits Conference*, The Hague, Netherlands, 1998. [Online]. Available: citeseer.nj.nec.com/278712.html
- [4] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A wireless sensor network for structural monitoring," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press, 2004, pp. 13–24.
- [5] G. Lu, N. Sadagopan, B. Krishnamachari, and A. Goel, "Delay efficient sleep scheduling in wireless sensor networks," in *Proc. of Infocom 2005*, Mar. 2005.
- [6] M. L. Sichitiu and C. Veerarittiphan, "Simple, accurate time synchronization for wireless sensor networks," in *IEEE Wireless Communications and Networking Conference (WCNC 2003)*, New Orleans, LA, Mar. 2003.
- [7] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press, 2004, pp. 39–49.
- [8] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Trans. Communications*, vol. 39, no. 10, pp. 1482–1493, Oct. 1991.
- [9] —, "Improved algorithms for synchronizing computer network clocks," in *Proc. of ACM*

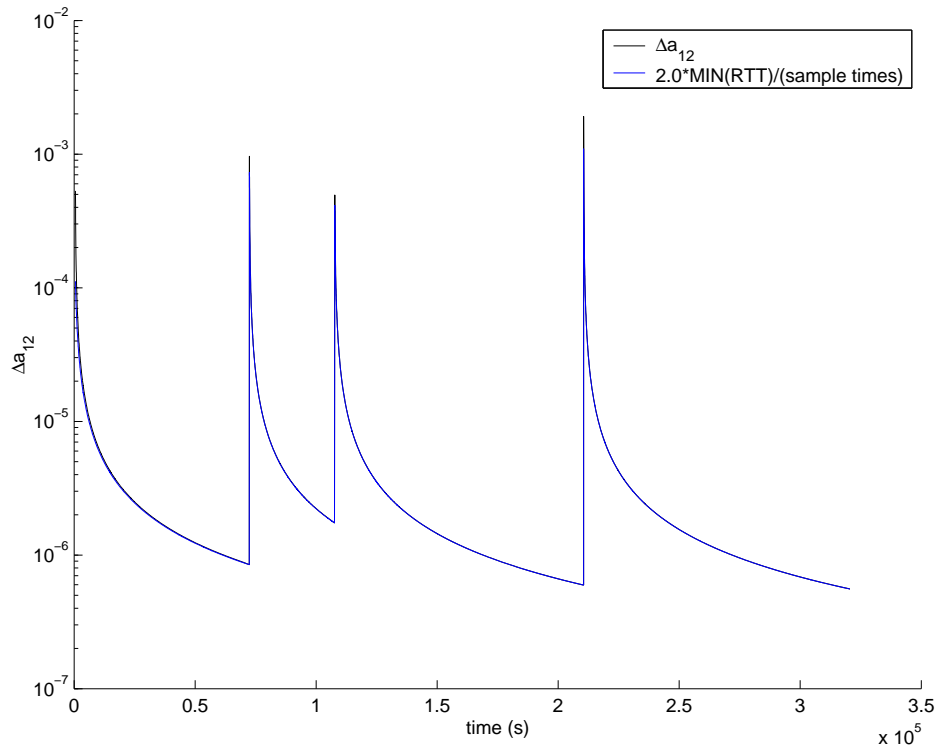


Fig. 13. The evolution of the bound on the relative drift Δa_{12} and the predicted evolution (18) for the second experiment.

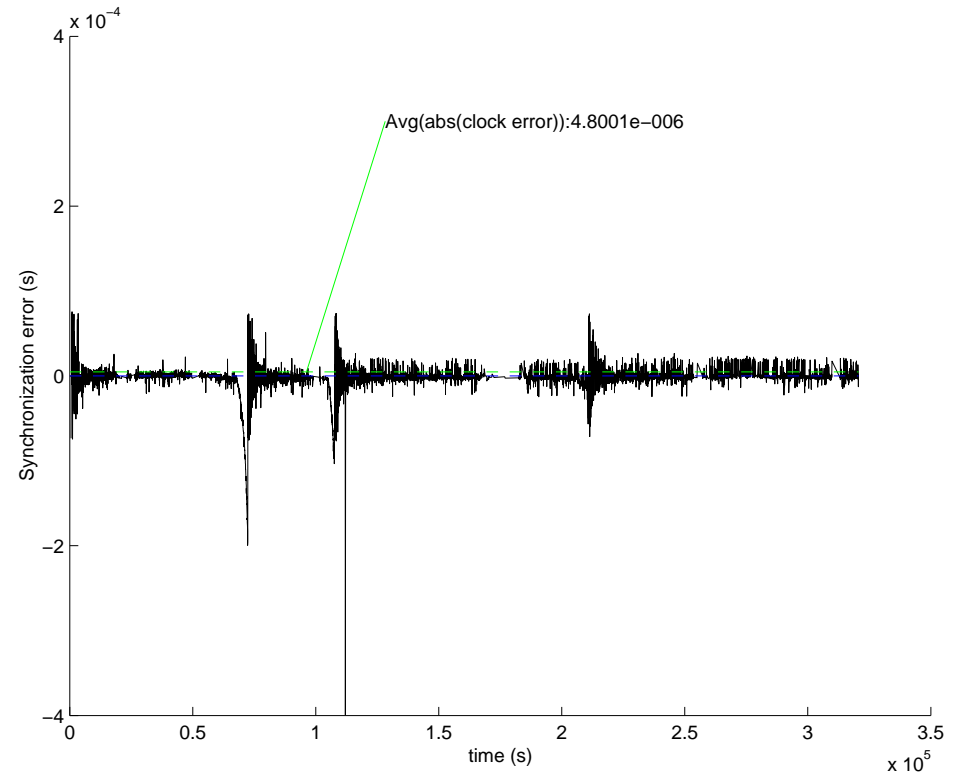


Fig. 14. The synchronization error for the second experiment.

Conference on Communication Architectures (ACM SIGCOMM'94), London, UK, Aug. 1994.

- [10] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton, "Sensor network-based countersniper system," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM Press, 2004.
- [11] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: A survey," *IEEE Network*, vol. 18, pp. 45–50, July 2004.
- [12] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization in wireless sensor networks: A survey," *Ad-Hoc Networks*, vol. 3, pp. 281–323, May 2005.
- [13] 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.
- [14] 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

- [15] K. Römer, "Time synchronization in ad hoc networks," in *Proc. of ACM Mobihoc*, Long Beach, CA, 2001.
- [16] L. Meier, P. Blum, and L. Thiele, "Internal synchronization of drift-constraint clocks in ad-hoc sensor networks," in *MobiHoc*, 2004.
- [17] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proc. of the First ACM Conference on Embedded Networked Sensor System (SenSys)*, Nov. 2003, pp. 138–149.
- [18] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System architecture directions for networked sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104. [Online]. Available: citeseer.nj.nec.com/382595.html
- [19] J. V. Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *WSNA'03*, 2003.
- [20] Q. Li and D. Rus, "Global clock synchronization in sensor network," in *INFOCOM*, 2004.
- [21] H. Dai and R. Han, "Tsync: a lightweight bidirectional time synchronization service for

wireless sensor networks,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, no. 1, pp. 125–139, 2004.

- [22] Crossbow, “Mica2 wireless measurement system,” <http://www.xbow.com/Products/productsdetails.aspx?sid=72>.
- [23] 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.
- [24] J. Polastre, R. Szewczyk, and D. Culler, “Telos: Enabling ultra-low power wireless research,” in *In Proceedings of IPSN/SPOTS*, Los Angeles, CA, Apr. 2005.
- [25] “Chipcon CC1000 transceiver.” [Online]. Available: <http://www.chipcon.com>

Suyoung Yoon received the B.S. degree in department of computer science and statistics from Seoul National University, Korea, in 1987 and the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology, in 1991. He is currently working towards the Ph.D. in the Department of Electrical and Computer Engineering at North Carolina State University. His advisor is Dr. Mihail L. Sichitiu and he is a member of WALAN research lab. Suyoung Yoon’s research interest includes wireless networking including ad-hoc and wireless sensor networking.

Mihail L. Sichitiu was born in Bucharest, Romania. He received a B.E. and an M.S. in Electrical Engineering from the Polytechnic University of Bucharest in 1995 and 1996 respectively. In May 2001, he received a Ph.D. degree in Electrical Engineering from the University of Notre Dame. He is currently employed as an assistant professor in the Department of Electrical and Computer Engineering at North Carolina State University. His primary research interest is in Wireless Networking with emphasis on ad hoc networking and wireless local area networks.