Symmetric Clock Synchronization in Sensor Networks

Philipp Sommer
Computer Engineering and
Networks Laboratory
ETH Zurich
8092 Zurich, Switzerland
sommer@tik.ee.ethz.ch

Roger Wattenhofer
Computer Engineering and
Networks Laboratory
ETH Zurich
8092 Zurich, Switzerland
wattenhofer@tik.ee.ethz.ch

Abstract

In this paper we argue that achieving symmetric errors is the key to an improved understanding of clock synchronization. We present a clock synchronization algorithm with drift compensation that implements this symmetric error paradigm. The performance of the algorithm is evaluated by measurements in an indoor testbed using the TinyNode hardware platform. We show that the remaining error is symmetric and in the range of the clock granularity.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Experimentation, Performance, Measurements

Keywords

Sensor Networks, Time Synchronization, Clock Drift, Symmetric Error, Energy Efficiency

1 Introduction

Clock synchronization is a major building block in wireless sensor networks. On the one hand, accurate clock synchronization is important for time-critical data fusion tasks. Only with accurately synchronized clocks one may get a global picture of an event that is sensed by several nodes. A prototypical application is the distributed sensing of an audio signal where precisely synchronized clocks may even empower the sensor network to guesstimate the position of an event, e.g. [11]. On the other hand, and maybe even more importantly, clock synchronization plays a major role in energy efficiency. State-of-the art energy-efficient sensor network protocols, e.g. [2], have advanced duty cycling schemes. Nodes are only awake during sensing and communication, if two nodes plan to communicate they will establish a rendezvous scheme. The better the clock synchronization, the less energy is wasted in the necessary guard times to not miss the rendezvous point.

We believe that clock synchronization deserves more research, despite the rich body of existing work, and despite the well-accepted protocols that are often taught in sensor network lectures, e.g. RBS [5] and TPSN [6]. Existing advanced clock synchronization protocols try to approximate the clock drifts using various heuristics. In this paper we ask ourselves the question: When should one be satisfied with a clock synchronization algorithm? Can one argue about the optimality of a clock synchronization algorithm? Towards this goal we treat clock synchronization like a guessing game: We try to synchronize our clock with a reference clock by exchanging messages. Naturally both our clock and the reference clock may experience offset and drift, changing over time. When sensing an event, our clock essentially asks itself: Which timestamp would the reference clock report? Clearly, because no hardware is perfect we cannot hope to perfectly answer that question. Instead, we try to answer it as well as possible, such that errors are minimized, which turns out to be the same as that positive and negative errors are in an equilibrium. In the paper we will argue that this approach leads to the best possible clock synchronization.

Unfortunately, clock synchronization algorithms are tested on hardware providing different clock granularity which makes it difficult to fairly compare results. A small variance in the synchronization error can be more important than a very small average error, e.g. when one wants to reduce the necessary guard times in duty cycling schemes. We argue that an evaluation of an algorithm should include both the error in absolute time and in relation to the clock granularity of the employed hardware system.

2 Related Work

Time synchronization has been studied extensively for distributed systems. Traditional time synchronization algorithms like NTP [8] are due to their nature not well suited for sensor network applications. GPS enabled devices provide an accurate time synchronization to UTC but are costly, energy consuming and limited to outdoor applications.

As research in sensor networks evolved during the last years, many different approaches for time synchronization were proposed ([9], [13], [10], [3], [7]). Due to space restrictions, we only compare to the most known time synchro-

REALWSN'08, April 1, 2008, Glasgow, United Kingdom.

nization protocols for wireless sensor networks: Reference Broadcast Synchronization (RBS) [5] and Timing-sync Protocol for Sensor Networks (TPSN) [6].

RBS exploits the broadcast nature of the physical channel to synchronize a set of receivers with one another. The timestamp of the reception of a broadcast message is recorded at each node and these timestamps are exchanged to calculate relative clock offsets between nodes. The RBS algorithm achieves a synchronization error of 11µs when running on Berkeley Motes. Given the 2µs resolution of the hardware clock, this corresponds to an average error of 5.5 clock ticks.

The TPSN algorithm builds a spanning tree of the network during the level discovery phase. In the synchronization phase of the algorithm, nodes synchronize to their parent in the tree by a two-way message exchange. The authors of [6] showed by simulations using Mica motes that TPSN achieves an average single-hop synchronization error of 16.9µs which is equal to 67.6 clock ticks given the 0.25µs clock resolution of the hardware.

Unfortunately, the papers [5] and [6] only present plots with absolute errors which makes it difficult to evaluate their protocols in terms of error symmetry. Because of Figure 5 in [6] we believe that the errors in RBS and TPSN are not symmetric.

In contrast our clock synchronization algorithm which concentrates on achieving a symmetric error offers accuracy in the order of a *single* clock tick. Although our software implementation utilizes a much slower clock source (32kHz), the remaining synchronization error is comparable to hardware platforms with finer clock granularity.

3 Implementation

This section describes a prototype implementation of a symmetric error clock synchronization algorithm on the TinyNode sensor platform.

3.1 Single-Hop Synchronization Algorithm

We employ the classical sender-receiver synchronization scheme to determine the message delay and the offset relative to the reference clock. The reference node periodically broadcasts a `SyncRequest` message based on the configured synchronization interval. Upon reception of this message, nodes wait for a random backoff time before they initiate the synchronization process by sending a `TimeSync` message to the reference node which answers with a `TimeSyncAck` message. If a node has not yet received an acknowledgment from the reference node after a specified time, it issues a new `TimeSync` message. Timestamping when the message is received or sent at the MAC layer eliminates the send, access and receive times (see [6]). Clocks start to run out of sync just after the synchronization point due to relative drift of the crystal oscillators used as the clock source. Depending on the requirements of the application, frequent resynchronization is necessary to keep the accuracy of the synchronization within the acceptable range. Drift compensation is indispensable to achieve high precision and low message overhead due to re-synchronizations. Heuristics like least squares linear regression [7] or recursive least squares estimation [12] were proposed to compensate for clock drift. We believe that these methods are susceptible to rounding errors which are hard to control.

In contrast, we show that even by using a simple moving average filter a symmetric synchronization error can be achieved. Based on the change of the clock offset during the synchronization interval, we estimate the relative drift to the reference clock. The weighted moving average filter smooths out the uncertainty in the measurement of the change of the offset:

$\Delta\text{offset}_{avg}(t) = \alpha \cdot \Delta\text{offset}(t) + (1-\alpha) \cdot \Delta\text{offset}_{avg}(t-1)$

The current timestamp of the reference clock can be extrapolated from the local timestamp using the offset determined at the most recent synchronization point and the estimated relative drift.

3.2 The Tinynode Platform

The TinyNode 584 sensor platform [4] features a MSP430 low-power microcontroller, 10 kB of RAM, 512 kB external flash memory and a XE1205 radio transceiver. The XE1205 radio module features low-power applications and offers data rates up 152 kbit/s. The MSP430F1611 microcontroller has two built-in 16-bit timers (Timer A and Timer B). The rate of Timer A can be configured as a fraction of the microcontroller clock rate (4MHz). Since the master clock of the microcontroller is disabled when the system enters low-power mode, Timer A cannot be used as a continuously running local clock. Being interested in energy-efficient applications mostly, we use Timer B as the local clock source. It is connected to an external 32kHz crystal oscillator and is operating also when the system is in low-power mode. The minimum clock granularity provided by Timer B is 30.5µs. Since the 16-bit timer will overflow every two seconds, we extend the timestamp to 32 bits using the 16 higher bits to count the number of counter overflows.

3.3 TinyOS Implementation

The implementation is done on top of the TinyOS operating system [1] which offers support for the TinyNode platform. However, a minor modification of the radio stack (`XE1205RadioM.nc`) was necessary to insert the current timestamp just before the packet is transmitted over the radio. The radio stack signals an event when the radio channel is clear for transmission. Then, the event handler method writes the current timestamp into the message payload. Right after that, the CRC checksum of the packet is calculated and the bytes are transferred to the 16-byte FIFO buffer of the radio module. It is assumed that the time consumed for the calculation of the checksum is deterministic and depends on the message size. Therefore, both message types used for the clock synchronization algorithm (`TimeSync` and `TimeSyncACK`) have the same size.

The radio module features a pattern detector which compares the received data stream with a predefined message preamble. If the received bits match the packet preamble, the subsequent bits are written into the 16-byte FIFO buffer. An interrupt is generated after the first byte has been transferred into the FIFO and the current timestamp is asssigned to the received message for further processing in the application layer.

4 Measurements

We performed a series of measurements on a indoor testbed with 10 TinyNode sensor nodes. Node 1 acts as the reference node, all other nodes are configured to synchronize their clocks periodically with the reference node. An additional node is connected through the serial port to a PC that controls the measurement procedure and overhears the radio traffic. At the beginning of each measurement run, all nodes are initialized by a sequence of configuration messages.

4.1 Clock Drift

In a first approximation, clock drifts are linear. We measured the relative clock drift of the TinyNodes used in our testbed. The current clock value of the nodes is probed using continuously broadcasted messages that act like external events. The reception of each message is timestamped with the local clock and written to the external flash memory. Figure 1 shows the measured offsets between the child nodes and the reference node (Node 1).

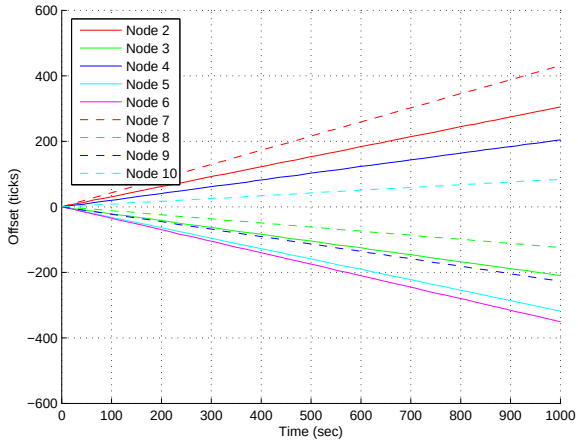

Figure 1. Relative drift with respect to Node 1. One can see the linear principal component.

What will remain if we remove the linear principal component from the clock drift? To get a better understanding of possible additional components of the clock drift we performed linear regression of the measured clock values. The distribution of the errors is shown in Figure 2. This secondary component of the clock drift is remarkably symmetric which motivated our approach.

4.2 Synchronization without Drift Compensation

In the next stage, we measured the performance of the synchronization algorithm with disabled drift compensation. The node connected to the PC continuously broadcasts messages of the type `EventMsg` that act like external events. Each `EventMsg` contains a sequence number to identify the event. The inter-arrival time between subsequent events is uniformly distributed between 10 and 24 seconds. Each node logs the estimated timestamp of the reference node together with the current local timestamp and the event identifier to the external flash memory. A single measurement run consists of approximately 1400 events.

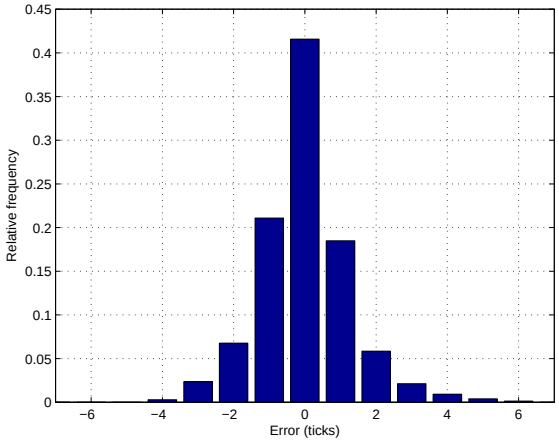

Figure 2. The distribution of errors for the best-fit line of the measured clock drift.

Figure 3 shows the measured error for a synchronization interval of 60 seconds. Due to the relative clock drift, nodes start to drift apart immediately after the synchronization point. The average synchronization error is 6.28 clock ticks (191.54µs) since the hardware clock of Node 3 runs slightly slower than the reference clock (see Figure 1). In addition, there is a strong correlation in the time series of the remaining error (Figure 4) since the absolute error is monotonically increasing after the synchronization point due to the relative drift.

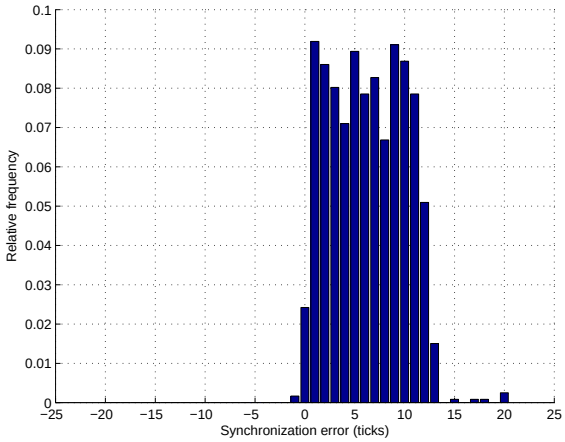

Figure 3. Distribution of the synchronization error between Node 3 and the reference node (without drift compensation).

4.3 Synchronization with Drift Compensation

Drift compensation significantly improves the accuracy of clock synchronization. We measured the performance of the drift compensation mechanism proposed in Section 3 (using $\alpha = 0.1$). The pair-wise synchronization error between Node 3 and the reference node is shown in Figure 5. The average synchronization error is reduced to 0.37 ticks (11.32 µs).

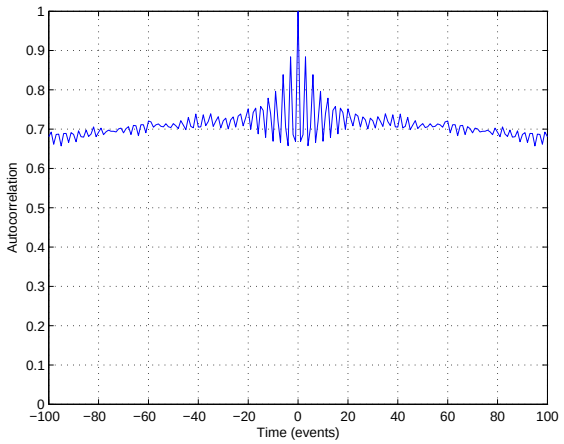

Figure 4. Autocorrelation of the synchronization error time series (Node 3, without drift compensation).

In the worst case, the clock value of the reference node is missed by a single clock tick. The distribution of the synchronization error is symmetric (Figure 6), indicating that positive and negative errors are in equilibrium. Nevertheless, a symmetric error alone is not a sufficient condition for optimality, since patterns in the error series might be exploited. To ensure that this is not the case, we compute the autocorrelation of the synchronization error time series (Figure 7). The plot of the autocorrelation shows nearly no dependency between the error of subsequent probes, in contrast to the strong correlation observed in Figure 4. Since no more patterns exist that can be exploited, we believe that our time synchronization algorithm is to some degree optimal given the constraints of the hardware platform.

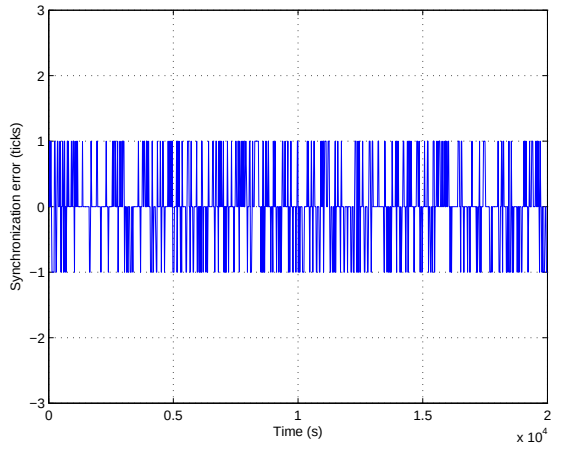

Figure 5. Synchronization error between Node 3 and the reference node.

A quick follow-up experiment using Mica2 motes was performed to investigate the influence of the clock granularity on the synchronization error. The implementation on the Mica2 platform offers a 912kHz hardware clock resulting in

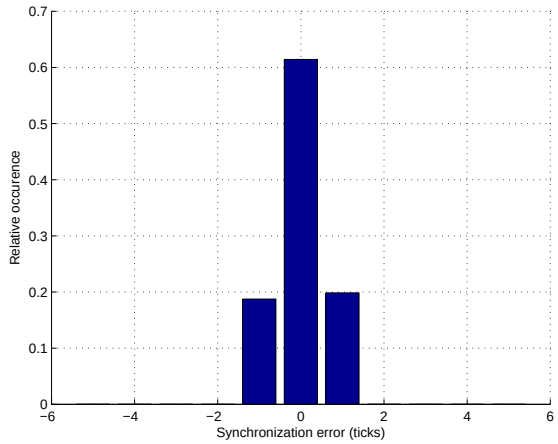

Figure 6. Distribution of the synchronization error between Node 3 and the reference node (with drift compensation).

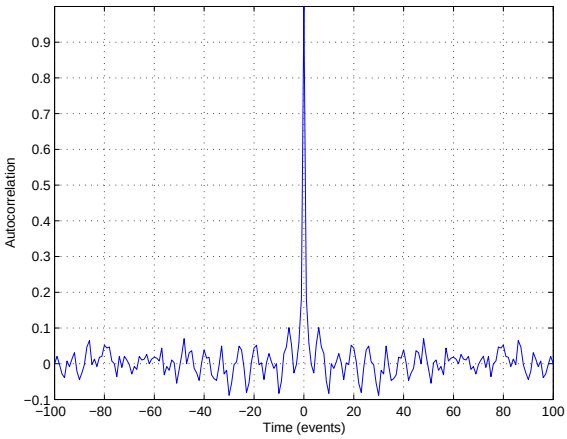

Figure 7. Autocorrelation of the synchronization error time series (Node 3, with drift compensation).

a granularity of around 1µs. Our early measurement results show that our algorithm achieves an average error of 1.46 ticks (1.59µs) and a standard deviation of 1.84 ticks (2.00µs) when synchronizing every 30 seconds. However, the distribution of the synchronization error still exhibits a small asymmetry. We believe that further fine-tuning can reduce the resulting synchronization error to the range of the clock granularity (1µs).

4.4 Energy Efficient Synchronization

Energy efficiency is an important aspect when designing applications for sensor networks. A deployed sensor node should ideally run with batteries for a long time period without the need of human interactions. Having the radio module switched on is very expensive compared to the energy consumption in low-power mode. The interval between clock synchronizations should be tuned in a manner that energy is saved while the accuracy of the synchronization remains still

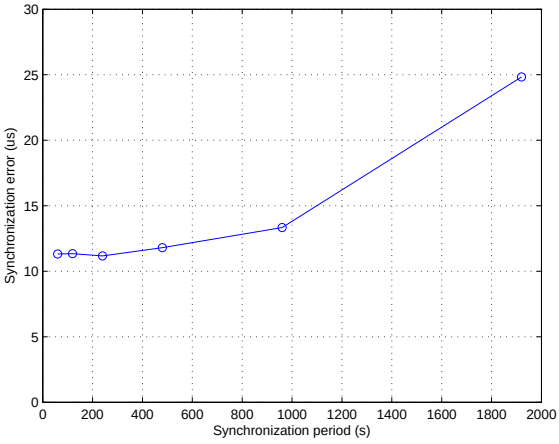

Figure 8. Average synchronization error depending on the synchronization interval.

within the acceptable range. A measurement series was performed to evaluate the impact of the synchronization interval on the accuracy of the synchronization. The initial synchronization interval was set to 60 seconds and doubled for each subsequent measurement run (see Table 1). The accuracy of clock synchronization degrades only slightly with increasing synchronization interval, as shown in Figure 8. The benefit in terms of accuracy is quite small when using a shorter synchronization interval than 960 seconds. Remarkable is the large increase of the synchronization error when doubling the synchronization interval from 960 to 1920 seconds.

	Error in ticks (µs)		
Interval	Average	Worst case	Std. dev
60s	0.37 (11.32)	2 (61.0)	0.61 (18.66)
120s	0.37 (11.34)	2 (61.0)	0.61 (18.69)
240s	0.36 (11.17)	2 (61.0)	0.61 (18.58)
480s	0.39 (11.79)	2 (61.0)	0.62 (19.03)
960s	0.44 (13.35)	3 (91.5)	0.69 (20.96)
1920s	0.81 (24.83)	5 (152.5)	0.83 (25.31)

Table 1. Measurement results for different synchronization intervals (1 tick = 30.5 µs).

5 Conclusion

Accurate synchronization of clocks is a prerequisite for many application in wireless sensor networks. In this paper, we presented a prototype implementation of a sender-receiver based single-hop clock synchronization algorithm. The performance of the algorithm was evaluated on an indoor testbed. Measurement results showed that the error between our estimation and the real reference clock value is very small (1-2 clock ticks). Furthermore, the remaining synchronization error is distributed in a symmetric manner and uncorrelated in time which we believe is optimal (see Section 4.3). A follow-up experiment on the Mica2 platform using a finer grained clock showed that our algorithm is competitive to existing algorithms in applications where small variances in the synchronization error are required.

6 References

[1] TinyOS. http://webs.cs.berkeley.edu/tos.

[2] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: Ultra-low power data gathering in sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, 2007.

[3] H. Dai and R. Han. Tsync: a lightweight bidirectional time synchronization service for wireless sensor networks. *SIGMOBILE Mob. Comput. Commun. Rev.*, 8(1), 2004.

[4] H. Dubois-Ferrière, L. Fabre, R. Meier, and P. Metrailler. Tinynode: a comprehensive platform for wireless sensor network applications. In *IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks*, 2006.

[5] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI), 2002.

[6] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, 2003.

[7] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004.

[8] D. Mills. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications*, 39(10), Oct 1991.

[9] K. Römer. Time synchronization in ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, 2001.

[10] M. Sichitiu and C. Veerarittiphan. Simple, accurate time synchronization for wireless sensor networks. In *WCNC '03: Proceedings of the IEEE Wireless Communications and Networking Conference*, 2003.

[11] G. Simon, M. Maróti, Á. 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*, 2004.

[12] R. Solis, V. Borkar, and P. Kumar. A new distributed time synchronization protocol for multihop wireless networks. *45th IEEE Conference on Decision and Control*, 2006.

[13] J. van Greunen and J. Rabaey. Lightweight time synchronization for sensor networks. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, 2003.