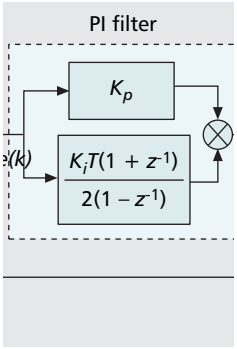
SELF-CORRECTING TIME SYNCHRONIZATION USING REFERENCE BROADCAST IN WIRELESS SENSOR NETWORK

FENGYUAN REN AND CHUANG LIN, TSINGHUA UNIVERSITY
FENG LIU, BEIHANG UNIVERSITY



Time synchronization is one of the most fundamental services for numerous wireless sensor network (WSN) applications. The authors introduce the definition and basic concepts of time synchronization, and summarize the related work.

ABSTRACT

Time synchronization is one of the most fundamental services for numerous wireless sensor network applications. In this article the definition and basic concepts of time synchronization are introduced, and the related work is summarized in brief. Through analyzing the characteristics of the existing typical synchronization protocols and making a comprehensive comparison of the performance of various algorithms, we present a common guideline for designing the time synchronization protocol in WSN. Following this guideline, we develop a new time synchronization protocol called Self-Correcting Time Synchronization (SCTS), which converts the time synchronization problem into an online dynamic self-adjusting optimizing process to make the offset compensation and drift compensation simultaneously. The time and space complexities of the algorithm implementation are very low. In addition, the SCTS protocol fully exploits the inherent broadcast property of wireless channel, so the communication overhead is rather low. Because the algorithm implementation is based on the phase locked loop principle, an equivalent digital PLL without an actual voltage controlled oscillator is also proposed to avoid introducing the extra hardware required by a traditional PLL circuit. Finally, we validate SCTS on the Berkeley Mica2 experimental platform, and the performance is evaluated and compared to the existing typical time synchronization protocol.

INTRODUCTION

Wireless sensor networks (WSNs) are attracting more and more attention from diverse research communities due to their potential usage in numerous commercial and military applications and scientific research, such as medical care, inventory tracking, battlefield surveillance, environment monitoring, and habitat monitoring. As in all distributed systems, time synchronization is very important in a sensor network since the design of many protocols and implementation of applications require precise time, for example, forming an energy-efficient radio schedule, conducting in-network processing (data fusion, data suppression, data reduction, etc.), distributing an acoustic beamforming array, and performing acoustic ranging (i.e., measuring the time of flight of sound), logging causal events during system debugging, and querying a distributed database.

Time synchronization is a research area with a very long history. Various mechanisms and algorithms have been proposed and extensively used over the past few decades. However, several unique characteristics of WSNs often preclude the use of the existing synchronization techniques in this domain. First, since the amount of energy available to battery-powered sensors is quite limited, time synchronization must be implemented in an energy-efficient way. Second, some messages need to be exchanged for achieving synchronization; however, the limited bandwidth discourages frequent message exchanges among sensors. Third, the small size of a sensor node imposes restrictions on computational power and storage space. Therefore, traditional synchronization schemes such as Network Time Protocol (NTP) and Global Positioning System (GPS) are not suitable for sensor networks because of complexity and energy issues, cost efficiency, limited size,and so on.

CLOCK MODEL AND SYNCHRONIZATION

Clock synchronization is of significant importance in WSNs. Before delving into the details of synchronizing clocks, we first define some terminologies used in this article. $c(t)$ denotes a *perfect clock*, where t is the *real time*, that is, coordinated universal time (UTC). Every sensor node maintains its own *local clock*, which is a monotonically nondecreasing function of t. This local clock is an ensemble of hardware and soft-

1536-1284/08/$25.00 © 2008 IEEE

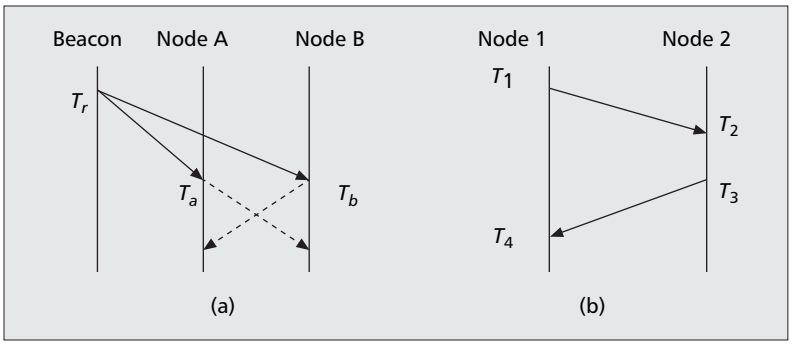


Figure 1. *Basic synchronization mechanisms: a) unidirectional broadcast; b) bidirectional pair-wise.*

ware components, essentially a timer that counts the oscillations of a quartz crystal running a particular frequency. In general, the timer is programmed to generate an interrupt, which is called a *clock tick*. At each clock tick, the interrupt procedure increments the clock value stored in memory.

For any two nodes' local clocks $c_i(t)$ and $c_k(t)$, the clock $c_i(t)$ is considered *correct* at time t if $c_i(t) = c(t)$, or the clock $c_i(t)$ is considered *accurate* at time t if $dc_i(t)/d_t = d_c(t)/d_t$, and two clocks $c_i(t)$ and $c_k(t)$ are *synchronized* at time t if $c_i(t) = c_k(t)$. The above definitions show that the two synchronized clocks are not always *correct* or *accurate*; time synchronization is not related to time correctness and accuracy. As for most applications in WSNs, it is sufficient to achieve clock synchronization. Since the oscillator frequency is time-varying due to ambient conditions such as temperature changes, variations of electric supply voltage, and air pressure, clock $c_i(t)$ of node i can be modeled as $c_i(t) = a_i(t)\ t + b_i(t_0)$, where $b_i(t_0)$ is the *clock offset,* the difference between the time reported by clock $c_i(t)$ and the *real time* at the initial instant t_0 (i.e., $b_i(t_0) = c_i(t_0) - c(t_0)$), and the *clock drift* $\rho_i(t)$ is defined as the difference in the frequencies of the clock $c_i(t)$ and the perfect clock $c(t) = t$ (i.e., $\rho_i(t) = dc_i(t)/dt - 1$). Thus, we also have $a_i(t) = \rho_i(t) + 1$. It is noted that $\rho_i(t)$ is a time variable, which reflects the random deviation of oscillator frequency from its nominal value. Straightforwardly, we can have

$$c_i(t) = a_{ik}(t)c_k(t) + b_{ik}(t_0), \qquad (1)$$

where $\rho_{ik}(t) = a_{ik}\ (t) - 1$ and $b_{ik}(t_0)$ are relative clock drift and relative clock offset, respectively.

A synchronization algorithm can either directly modify the local clock $c_i(t)$ or otherwise construct a *software clock* $h_i(t)$. A software clock is a function that takes a local clock value $c_i(t)$ as input and transforms it into $h_i(t)$. For example, $h_i(t) = h_i(t_0) + g\{c_i(t) - c_i(t_0)\}$ is a software clock that starts with the correct real time t_0, where function $g\{\cdot\}$ is used to convert the counter value of the local clock into the interval to construct one software clock. The popular WSN node Mica2 has a crystal frequency of 4 MHz, which means that the resolution of the local clock is 0.25 μs. If the software clock counter has an increment every 64 clock ticks, $g\{1\} = 16$ μs, which implies that the frequency of the software clock is 62.5 kHz, and its resolution is 16 μs.

Related Work

In the context of WSNs, time synchronization refers to the problem of synchronizing clocks across a set of sensor nodes that are connected to one another over single-hop or multihop wireless networks. Up to now, many protocols have been designed to address this problem, and typical algorithms include Reference Broadcast Synchronization (RBS) [1], Lightweight Time Synchronization (LTS) [2], TSync [3], Timing-Sync Protocol for Sensor Networks (TPSN) [4], Flooding Time Synchronization Protocol (FTSP) [5], and Tiny-Sync and Mini-Sync (TS/MS)[6]. These protocols all have some basic features in common: a simple connectionless messaging protocol, exchange of clock information among nodes, mitigating the effect of nondeterministic factors in message delivery, and processing utilizing different schemes and algorithms, respectively. They can be classified into two types: bidirectional pair-wise synchronization and unidirectional broadcast synchronization. TPSN, LTS, TSync, and TS/MS fall into the former group, RBS and FTSP the latter.

Unidirectional broadcast synchronization is also called receiver-receiver synchronization. A node periodically broadcasts wireless beacon messages to its neighbors. The receivers use the message's arrival time as a point of reference for comparing their clocks, and then exchange the local timestamps of when they received the same broadcast message, and finally compute their offset based on the difference in reception times to synchronize their clocks. This basic mechanism is shown in Fig. 1a. RBS is a typical receiver-receiver protocol. Compared to the traditional protocols working on a LAN, its main contribution is to directly remove two of the largest sources of nondeterminism involved in message transmission, transmission time and access time, through exploiting the concept of a *time-critical path*, which is the path of a message that contributes to nondeterministic synchronization errors. Therefore, RBS can provide a high degree of synchronization accuracy in sensor networks [1]. FTPS uses a fine-grained clock, medium access control (MAC) layer timestamping with several jitter reducing techniques, and clock drift estimation to achieve relatively high precision [2].

Bidirectional pair-wise synchronization is also called sender-receiver synchronization, and is performed by a handshake protocol between a pair of nodes. Figure 1b illustrates its fundamental mechanism, including three steps:

- Sender node A sends a message with its local time T_1 as a timestamp, and receiver node B receives this packet at its local time T_2, where $T_2 = T_1 + d + \delta$. Here, δ and d represent the offset between the two nodes and the end-to-end delay respectively.
- At time T_3, node B sends back an acknowledgment packet. This packet contains the values of T_2 and T_3. Node A receives the packet at T_4. Similarly, T_4 is related to T_3 as $T_4 = T_3 + d - \delta$.

• Sender node A can now calculate the clock offset and end-to-end delay as $d = [(T_2 - T_1) + (T4 - T_3)]/2$ and $\delta = [(T_2 - T_1) - (T_4 - T_3)]/2$.

Subsequently, sender node A can synchronize its clock to receiver node B's clock. Although TPSN, LTS, TSync, and TS/MS employ the same bidirectional pair-wise synchronization mechanism, they have exclusive features to meet the particular requirements of different applications. For example, to improve precision, TPSN takes advantage of the availability of the MAC layer. LTS provides a specified precision with little overhead, rather than striving for maximum precision. Both TS and MS use multiple pair-wise round-trip measurements and a line-fitting technique to obtain the offset and drift of the two nodes, rather than directly calculating the offset using the above equations. In addition, in TS/MS, once node B receives the message from node A, it needs to reply immediately without any delay. It is beneficial to reduce storage overhead through removing as many data points as possible before the line fitting. TS and MS only differ in this elimination step. TS uses a heuristic to keep only two data points for each of the two lines. However, the selected points may not be the optimal ones. MS uses a more complex approach to eliminate exactly those points. Hence, TS gives a suboptimal solution with minimal overhead, but MS provides an optimal solution with increased overhead. TSync has a centralized version, called the Hierarchical Referencing Time Synchronization (HRTS) protocol, and a decentralized version, called the Individual Time Request (ITR) protocol. Both protocols exploit the usage of dedicated radio channel for synchronization messages to avoid inaccuracies due to variable delays introduced by packet collisions.

Some other interesting time synchronization mechanisms and algorithms have also been proposed. For instance, Qun Li and Daniela Rus put forward Asynchronous Diffusion (AD) [7], in which any node can update its clock value asynchronously with respect to other nodes in the network through computing average clock readings. Hu and Servetto [8] defined a protocol for global synchronization in dense sensor networks. The clock synchronization proceeds in concentric waves, starting from a master node located in the center of the network. In the literature Hong and Scaglione [9] proposed a bio-inspired network synchronization protocol for large-scale sensor networks that emulates the simple strategies adopted by biological agents (e.g., fireflies).

Self-Correction Time Synchronization

In fact, Eq. (1) points out two basic synchronization principles: offset compensation and drift compensation. If the synchronization algorithm can determine the relative offset $b_{ik}(t_0)$ between node clock $c_i(t)$ and clock $c_k(t)$ at synchronization time t, clock $c_k(t)$ can be readily synchronized to clock $c_i(t)$ through compensating for the relative offset $b_{ik}(t_0)$ at a series of synchronization points. The corresponding synchronized clock $c_k^o(t)$ is depicted in Fig. 2. (For the sake of simplicity and clarity, we assume that both clocks $c_i(t)$ and $c_k(t)$ are not accurate, and have different drifts but with constant values; moreover, they have the identical initial value: $c_i(0) = c_k(0)$). Since the offset compensation does not take the impact of clock drift on synchronization precision into account, the synchronized clock $c_k^o(t)$ keeps the same varying rate with the local clock $c_k(t)$, which implies that the longer the synchronization interval, the larger the synchronization error. In order to improve precision we can decrease the synchronization period, but this will introduce too much overhead. If the relative clock drift $a_{ik}(t) = a_i(t) - a_k(t) = -a_{ki}(t)$ is accurately estimated, we can build a synchronized clock using drift compensation. $c_k^d(t)$ in Fig. 2 is such a synchronized clock whose frequency is independent of the local clock $c_k(t)$, but approximately approaches that of the reference clock $c_i(t)$. If the estimation of relative clock drift was unbiased, $c_k^d(t)$ and $c_i(t)$ would overlap (absolutely synchronize). However, in practice it is difficult to obtain unbiased estimation due to a variety of factors including short-term effects such as crystal frequency noise, temperature, and humidity changes, and long term effects such as crystal aging and noise in observations. We can only estimate it as accurately as possible. The more accurate this estimation value, the smaller the synchronization error in the longer period. Compared to offset compensation, which is effective for short-term time synchronization, drift compensation is beneficial for long-term synchronization. Naturally, if these two compensation techniques are employed together, the time synchronization algorithm with high resolution and moderate overhead can be yielded. The synchronized clock $c_k^{od}(t)$ shown in Fig. 2 intuitively demonstrates this judgment.

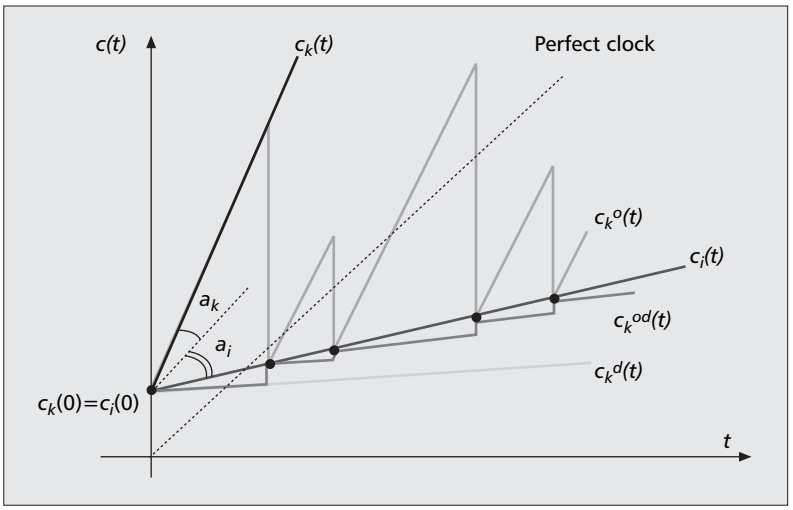


Figure 2. *Offset compensation and drift compensation.*

All aforementioned existing algorithms carry out the offset compensation, but only some algorithms utilize drift compensation, including RBS, FTSP, and TS/MS. The method widely used to estimate clock drift is linear regression, which requires a fixed number of historical synchronization points stored locally

The method used widely to estimate the clock drift is linear regression, which requires a fixed number of historical synchronization points stored locally in a table at each node.

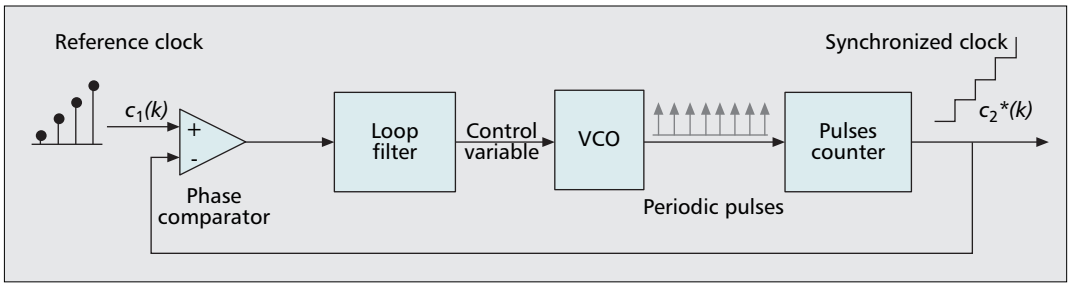


Figure 3. *The components of PLL for clock synchronization.*

in a table at each node. With more timing data, the least square estimator used for linear regression can generally make a better clock drift estimation. For example, Tiny-Sync only uses two timestamps, which means that the storage overhead is very small, but it hardly gives the optimal estimation value. On the contrary, Mini-Sync is able to provide the satisfied drift estimation, but it needs more than 40 data points. As for a sensor node with limited resources, it is unreasonable that the space complexity of any employed algorithms be too high. In addition, least square regression also requires significant computation. Can we find other algorithms with low space and time complexity to estimate the clock drift? It is one of the motivations in this work. On the other hand, although bidirectional pair-wise synchronization algorithms can achieve the goal of time synchronization across the whole network, they need many message exchanges among nodes, which consumes too much energy, which is likely to shorten network lifetime. For instance, there are n nodes in a cluster. At least $2(n-1)$ message exchanges are required to synchronize all clocks in the cluster. The beacon node sends $n-1$ messages and receives $n-1$ messages. Each node sends one message and receives one message. As opposed to two-way synchronization, unidirectional broadcast synchronization can also provide comparative global time in the network with low communication overhead since it sufficiently exploits the broadcast property of the wireless communication medium. FTSP is a good example. It is noted that RSB is a typical broadcast synchronization protocol, but only makes use of broadcast property to eliminate possible errors on the sender side, and does not take reducing communication overhead into consideration because receivers need to exchange timestamps. Assume that there are n nodes in a broadcast domain. A number of $2n$ exchange messages is needed when the timestamps are collected and evaluated at a central node. In scenarios without central nodes, exchanging timestamps takes $n(n-1)$ packets. Another flaw of FTSP is that it requires calibration on the hardware actually used in the deployment; it is not a pure software solution independent of hardware. The comparatively high time and space complexity of the linear regression algorithm used by FTSP is another flaw. Summarizing the features of the existing algorithms, we believe it should be preferable to revise a time synchronization scheme that adopts offset compensation and drift compensation simultaneously; also, both time complexity and space complexity of the corresponding algorithm are very low, and the communication overhead is also small. Subsequently, we propose a new synchronization scheme based on this basic understanding and idea.

First, in order to reduce communication overhead, the inherent broadcast property should still be fully exerted. The beacon node periodically broadcasts the reference timestamped packets at the physical layer. Since short-range wireless communication is used in WSNs, the coverage range of the broadcast domain is very limited; for example, the maximum transmission distance defined in IEEE 802.15.4 is 30 m, and the corresponding maximum propagation delay is only 100 ns, which does not impose any perceptible impact on the synchronization precision specified to be on the order of 1 μs. In other words, the synchronization error caused by the propagation delay can be negligible. Different from the existing broadcast synchronization scheme, in which once the receiver timestamps a broadcast packet, the synchronization process is immediately executed based on this individual message), our new scheme uses a sequence of successive broadcast reference packets to drive the synchronized clocks on the receiver nodes to gradually approach and eventually be locked to the reference clock on the beacon node. By this means, all the clocks in a broadcast domain will be successfully synchronized.

To realize this synchronization solution, we need to employ a phase locked loop (PLL) algorithm, which is a well-known synchronization algorithm found in many synchronization schemes (NTP etc.). Its advantage is the low cost of its implementation, together with generally acceptable performance. Schematically, a PLL is arranged as in Fig. 3. It contains three main components: a loop filter, a voltage controlled oscillator (VCO), and a pulse counter. The VCO and counter constitute the local clock hardware. The error between the reference clock and the local clock enters the loop filter, which is in charge of eliminating possible noise. The output of the filter controls the frequency of the VCO and eventually the local clock itself.

Generally, the PLLs employed for clock synchronization are digital; thus, we can describe it using the block diagram in the discrete domain shown in Fig. 4a. For the sake of convenience, the loop filter is defined as a proportional integrative (PI) filter, whose parameters are K_p and K_i. T is the synchronization period. $c_1(k)$ denotes

the timestamp of the reference clock carried in the broadcast packet, and $c_2^*(k)$ is the value of the synchronized clock on an arbitrary node. The VCO is necessary for a basic PLL algorithm and is an actual device. Taking the requirement of reducing cost into account, we do not expect to introduce extra hardware cost only for time synchronization. Next, we give an equivalent scheme of a digital PLL that can directly work on the local crystal oscillator instead of operating on an actual VCO.

From Fig. 4a, we can easily obtain

$$c_2^*(k+1) = c_2^*(k) + K_0 v(k) T, \quad (2)$$

where T is synchronization period: $T = t(k+1) - t(k)$; here, t is the real time. Since the nominal frequency of VCO K_0 is constant and free running, we can replace it with the intrinsic frequency of the crystal oscillator on the local node. Since the counter simply counts the number of pulses between the two broadcast packet arrivals,

$$K_0\{t(k+1) - t(k)\} = c_2(k+1) - c_2(k), \quad (3)$$

where $c_2(k)$ is the value of counter of the local clock. Substituting Eq. 3 into Eq. 2 yields

$$c_2^*(k+1) = c_2^*(k) + v(k)[c_2(k+1) - c_2(k)]. \quad (4)$$

So far, we obtain an equivalent expression of a digital PLL without an actual VCO. Figure 4a illustrates its structure, which is useful for implementing a software PLL. Functionally, it is the same as that in Fig. 4b.

To determine the values of parameters K_p and K_i, we use the zero-pole assignment method [10] to tune the PI filter. From Fig. 4, we readily obtain the open-loop transfer function and the characteristic equation of closed-loop system, respectively:

$$G_o(z) = \frac{K_{0T}\lfloor (2K_p + K_i T)z + K_i T - 2K_p \rfloor}{2(z-1)^2} \quad (5)$$

$$2(z-1)^2 + K_0 T(2K_p + K_i T)z + K_0 T(K_i T - 2K_p). \quad (6)$$

Given that $z = 0.5$ is a zero point of the open-loop system and there are two identical poles in the closed-loop system. Through some derivation and simplification, we have the following parameter relationship:

$$2K_p = 3K_i T \quad (7)$$

$$K_i K_0 T_2 = 1 \quad (8)$$

The inherent frequency of the crystal oscillator of Mica2 mote K_0 is 62.5 kHz. Let synchronization period T equal 1 s; then

$$K_i = 1.6 \times 10^{-5}, K_p = 2.4 \times 10^{-5}.$$

Our new time synchronization scheme fully makes use of the broadcast property of WSNs; each node independently adjusts its synchronized clock according to the reference clock without any message exchange among nodes.

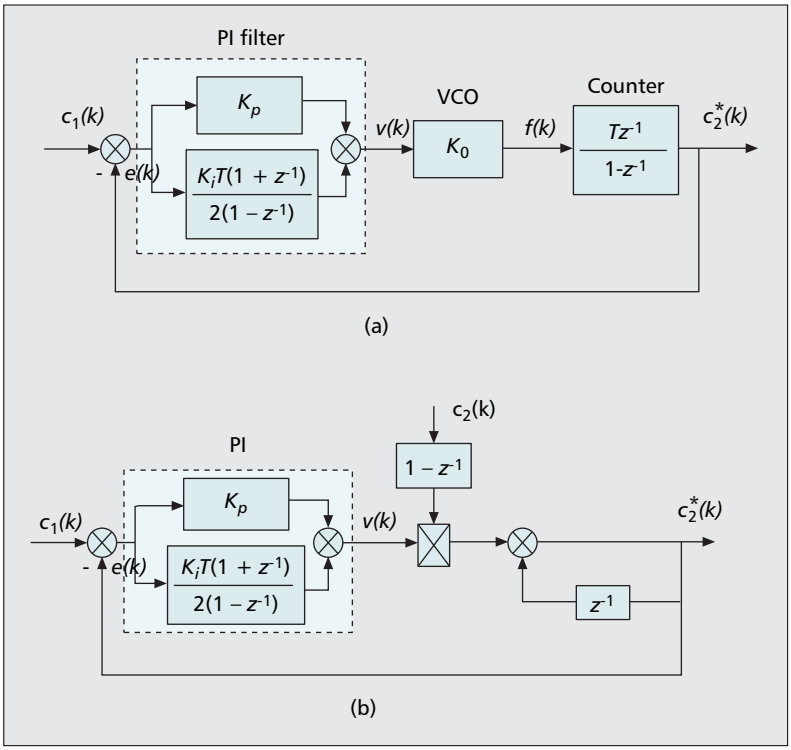


Figure 4. *The digital phase locked loop: a) the block diagram of a digital PLL; b) an equivalent of the digital PLL without VCO.*

Hence, it is also called the Self-Correcting Time Synchronization (SCTS) scheme. Besides saving energy due to low communication loads, SCTS has the following advantages:

- Merging the offset compensation and drift compensation into an online dynamic self-adjusting optimizing process.
- Low space complexity since the history timestamp data are compressed into a variable $e(k-1)$ to provide contributions for drift estimation.
- Low time complexity since there are no any complex matrix computations used by the least-square linear regression.
- Capable of absorbing some noise caused by the uncertain factors occurring during time stamping the reference broadcast packets on the sender and receiver sides. Essentially, it is the low-pass PI filter that can eliminate most of the high-frequency components in the sequence of reference clocks, which is beneficial to improve precision. Of course, we can also use the MAC layer timestamping and other jitter reducing techniques to achieve the higher precision.
- Robust against reference clock signal loss.

Naturally, the more frequent the arrivals of the reference signal, the better the accuracy of synchronization (i.e., the error is smaller). This feature is advantageous in designing a proactive time synchronization protocol for long-running applications. In other words, the precision of SCTS can be arbitrarily defined through changing the synchronization period. On the other hand, the SCTS scheme has a transient self-adjusting process and employs a narrow bandwidth filter; thus, it has a drawback in having relatively slow convergence of the local synchro-

Time synchronization is an indispensable part of infrastructure in wireless sensor networks. We conclude that an effective time synchronization scheme should consider both the offset compensation and drift compensation simultaneously.

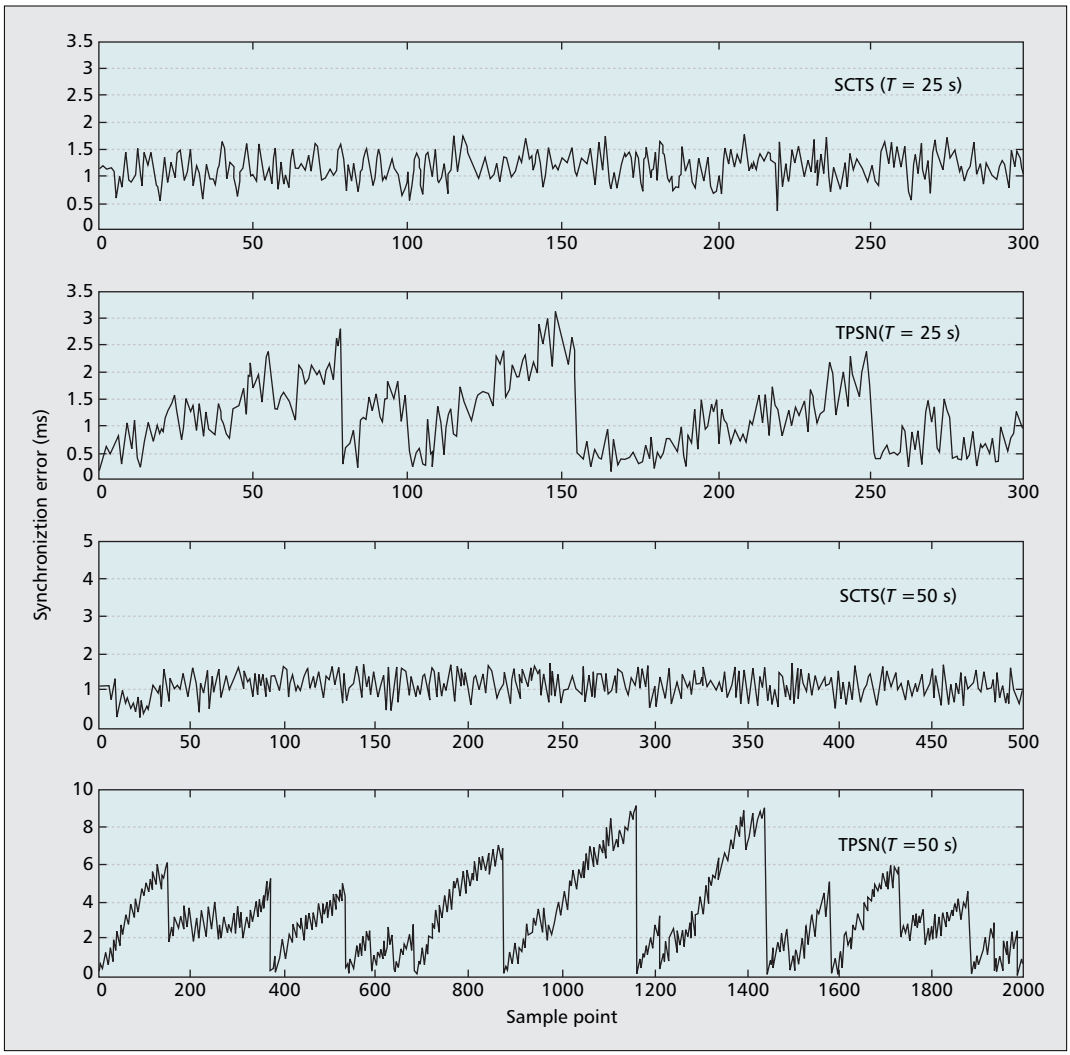


Figure 5. *Synchronization error.*

nized clock to the reference clock. However, it is acceptable for a proactive time synchronization protocol suitable for many WSN applications.

Experiment Results

To validate the SCTS algorithm and evaluate its performance, we implemented it on the popular Berkeley Mica2 mote and built a small-scale network, which consists of 10 nodes in a single broadcast domain. The node mounted on the MIB510 board acts as not only the beacon node but also the sink node, and periodically broadcasts the reference clock signal. To measure the synchronization precision, the sink node broadcasts a query for the counter value of all local synchronized clocks in the middle of two successive reference clock broadcasts. Once receiving the request, each node records the current counter value of the synchronized clock and then returns it to the sink node. After collecting the messages from all nodes, the sink node hands it over to the PC machine via the *SerialForwarder* program built in TinyOS. The synchronization error is defined as the maximum offset between the reference clock and all synchronized clocks. This is not a perfect solution because it may introduce measurement errors. For convenience in clarifying our argument and making a comparison, we use the same experiment configuration and measurement approach to evaluate the TSPN algorithm. Setting the period T to 25 s and 50 s, we conduct the experiments and draw the sampling points of synchronization errors in Fig. 5. Obviously, the relative clock drift severely deteriorates the performance of the synchronization scheme merely adopting offset compensation (like TPSN etc.). The drift compensation mechanism employed by some algorithms (our SCTS etc.) makes a considerable improvement. Although the performance is related to the synchronization frequency, the precision is not too sensitive to the frequency variance when it is limited in a certain range because the crystal oscillator approximately operates in a stable status so there is no drastic change in ambient conditions. To clearly demonstrate the performance improvement, we compute the statistics of synchronization errors and list them in Table 1.

Conclusions and Future Work

Time synchronization is an indispensable part of the infrastructure in wireless sensor networks. We conclude that an effective time synchroniza-

tion scheme should consider both offset compensation and drift compensation simultaneously; also, both time complexity and space complexity of the algorithm should not be very high, and the communication overhead should also be limited. To achieve this goal, in this article we convert the clock synchronization problem into an online dynamic self-adjusting optimizing process. The implementation algorithm based on PLL can make offset and drift compensations with low time and space complexity at the cost of slow convergence. The measurement results from the experiments on Berkeley motes show that our SCTS scheme outperforms TPSN with tthe same configurations and measurement methods. Although our current investigation is limited to a single broadcast domain, the proposed SCTC algorithm is readily extensible to network-wide time synchronization using the mechanism of root node discovery and maintenance revised in the FTSP algorithm [5]. This will be a subject of our future work. In addition, although the measurement methods are sufficient to make the verification, they are not accurate enough. We will design a precise measurement approach, for example, equipping a GPS receiver for every node, to carefully evaluate the performance and determine the achievable precision of various algorithms in future work.

Acknowledgments

This work is supported in part by a grant from the National Grand Fundamental Research 973 Program of China (2006CB303000), National Natural Science Foundation of China (No. 60573122 and 60773138), and National High-Tech Research and Development Plan (863) of China (No. 2006AA01Z225, 2006AA01Z223 and 2006AA09Z117). The authors would like to thank editors and anonymous reviewers for their insightful comments.

References

[1] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts," *Proc. 5th Symp. Op. Sys. Design and Implementation*, 2002, pp. 147–63.
[2] J. Van Greunen and J. Rabaey, "Lightweight Time Synchronization for Sensor Networks," *Proc. 2nd ACM Int'l. Wksp. Wireless Sensor Networks and Apps.*, 2003, pp. 11–19.
[3] H. Dai and R. Han, "Tsync: A Lightweight Bidirectional Time Synchronization Service for Wireless Sensor Networks," *ACM SIGMOBILE Mobile Computing and Commun. Rev.*, vol. 8, no. 1, 2004, pp. 125–39.
[4] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-Sync Protocol for Sensor Networks," *Proc. 1st ACM Conf. Embedded Networked Sensor Sys.,* 2003, pp. 138–49.

Item	SCTS		TPSN	
	T = 25 s	T = 50 s	T = 25 s	T = 50 s
Mean (ms)	1.162	1.126	1.28	3.07
Deviation (ms)	0.282	0.316	0.53	4.00
Maximum (ms)	1.760	1.728	3.86	9.07

Table 1. *Statistic of synchronization error.*

[5] M. Maroti *et al.*, "The Flooding Time Synchronization Protocol," *Proc. 2nd Int'l. Conf. Embedded Networked Sensor Sys.*, 2004, 39–49.
[6] M. L. Sichitiu and V. C. Simple, "Accurate Time Synchronization for Wireless Sensor Networks," *Proc. IEEE Wireless Commun. and Networking*, 2003.
[7] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," *Proc. IEEE INFOCOM '04*, Hong Kong, 2004.
[8] A.-S. Hu and S. D. Servetto, "Asymptotically Optimal Time Synchronization in Dense Sensor Networks," *Proc. 2nd ACM Int'l. Wksp. Wireless Sensor Networks and Apps.*, Sept. 2003, pp. 1–10.
[9] Y. W. Hong and A. Scaglione, "A Scalable Synchronization Protocol for Large Scale Sensor Networks and Its Applications," *IEEE JSAC*, vol. 23, no. 5, 2005, pp. 1085–98.
[10] B. Friedland, *Advanced Control System Design*, Prentice Hall, 1996.

Biographies

Fengyuan Ren (renfy@csnet1.cs.tsinghua.edu.cn) is an associate professor at Tsingua University. He obtained a Ph.D. degree in computer science from Northwestern Polytechnic University in 1999. From 2000 to 2001 he worked in the Electronic Engineering Department of Tsinghua University as a postdoctoral fellow. In 2002 he moved to the Computer Science and Technology Department of Tsinghua University. He has more than 60 papers in journals and conferences. His research interests include flow control in computer networks, control in/over networks, wireless networks, and wireless sensor networks.

Chuang Lin is a professor with the Department of Computer Science and Technology, Tsinghua University. He received a Ph.D. degree in computer science from Tsinghua University in 1994. His current research interests include computer networks, performance evaluation, network security analysis, and Petri net theory and its applications. He has published more than 200 papers in leading journals and conference proceedings in these areas, and has published three books.

Feng Liu received a Ph.D. degree in control science and engineering from Xi'an Jiaotong University, China, in 2000. From September 2000 to December 200 he was a postdoctoral fellow in the Department of Electronics Engineering, Tsinghua University. Now he is an associate professor with the School of Electronic and Information Engineering, Beihang University. His research interests include complex networks and systems, delay- and disruption-tolerant networking, and transmission control protocols of satellite networks.