Implementation of High Precision Synchronization Protocols in Wireless Sensor Networks

Djamel Djenouri, Miloud Bagaa
CERIST Research Center, Algiers, Algeria
ddjenouri@mail.cerist.dz, bagaa@mail.cerist.dz

***Abstract*—Microsecond-level time synchronization is needed in realtime applications of wireless sensor networks. While several synchronization protocols have been proposed, most performance evaluations have been limited to theoretical analysis and simulation, with a high level of abstraction by ignoring several practical aspects, e.g. packet handling jitters, clock drifting, packet loss, etc. Effective implementation in real motes faces several challenges due to motes' limitations and the unreliable lossy channels. These issues affect the protocol performance and precision. Authors of some pragmatic solutions followed empirical approaches for the evaluation, where the proposed solutions have been implemented on real motes and evaluated in testbed experiments. This article throws light on issues related to the implementation of synchronization protocols in WSN. The challenges related to WSN environment are presented, the importance of real implementation and the testbed evaluation are motivated by some experiments that we conducted. Finally, some relevant implementations of the literature that meet microsecond-level precision are discussed.**

I. Introduction

Fine-grained time synchronization is essential for realtime applications of wireless sensor networks (WSN); e.g., nuclear or strategic center monitoring evolving realtime actuation, disaster relief operations, health-care and telemedicine using in-vivo sensors/actuators, etc. Consequently, several synchronization protocols have been proposed in the literature. Most evaluation of these protocols are based on network simulations for comparison with state-of-the-art candidates, as well as numerical analysis for the evaluation of estimators and possible comparison of the mean square errors (MSE), or its variants, with a theoretical optimum, e.g. Cramer-Rao lower-bound (CRLB) [1]. The importance of such approaches for getting a preliminary vision on a protocol performance, or investigating issues such as scalability, cannot be neglected. However, they are far from being effective substitute of testbed experimentation, as many aspects are either neglected, or simulated with ideal assumptions at a high level of abstraction. For instance, delays and jitters are assumed to ideally follow some distribution (e.g. Gaussian), if not neglected, clock drifting is not thoroughly modeled, packet loss is seldom considered, etc. Empirical evaluation is more pragmatic and allows to have a concrete view on the synchronization protocol, its features and limitations. Existing implementations can be reused in other applications or by other protocols requiring time synchronization. Therefore, having a horizontal vision on current implementations is essential to decide which implementation can be adequately reused to fulfil one application's requirements or another, or which one can be adapted with minimum amendments. The aim of this article is to throw some light on issues related to implementation of a synchronization protocol into sensor motes, to present and discuss state-of-the-art implementations. Only implementations allowing for a fine-grained precision at the microsecond level are considered. Those that use clocks with low-resolution that do not permit microsecond granularity are not considered.

The rest of the article is organized as follows. The related work is summarized in the next section, followed by the implementation challenges in Section III with some experimental illustrations. Section IV presents our investigation on the impact of some empirical parameters. Some implemented protocols are presented in Section V. This presentation is focusing on implementation aspects and is limited to a selection of representative protocols, and it is far from targeting an exhaustive review. Final, Section VI concludes the article.

II. Related Work

Some survey articles on time synchronization in wireless sensor networks have been published in the last few years. Three synchronization categories have been reported by Sivrikaya and Yener [2], i) event ordering, as the simplest form of synchronization, ii) synchronization to a reference, and iii) relative clocks. In the latter, each node runs its local clock independently but maintains information about relative skew and offset to other nodes for possible conversion. Most of the synchronization solutions proposed for WSN use this model. Sundararaman et al. [3] give a detailed survey with more taxonomy on existing solutions, up to 2005. Several classifications have been proposed. The first classification considers what the authors called the synchronization issues. Protocols are divided into i) master-slave vs. Peer2peer, ii) clock correction vs. clock untethered, iii) Internal vs. external, iv) probabilistic vs. deterministic, and v) sender-to-receiver vs. receiver-to-receiver. The second classification considers application dependent issues. Protocols are split up into, i) single-hop vs. multi-hop, ii) stationary vs. mobile networks, iii) MAC layer based vs. standard approach. In the MAC layer approach, the MAC protocol is used to encapsulate and timestamp synchronization messages.

Another interesting recent survey article is by Wu et al. [1]. The authors focus on signal processing and theoretical issues of the synchronization, where the solutions are presented from the perspective of the mathematical methods for estimation

of synchronization parameters and theoretical analysis. A similar survey from the empirical and practical perspective is missing in the current literature. This represents the aim of this manuscript, where the real implementations of the protocols are analyzed, relevant issues are presented and discussed.

III. Implementation Challenges

In this section, the different challenges that one faces when designing, implementing, and testing a synchronization protocol are presented. First and foremost, the fast drifting of clocks used by motes and mote limitations are inevitable results that results from reducing the sensor mote cost and size, for economic benefits. Such reduction comes at the use of memory and computation limited micro-controllers, cheap but fast drifting clocks. Delay variation is a feature inherited from the wireless lossy environment, to which add long jitters for in-node message processing at sensor nodes that is caused by the mote limitation and instability. All these challenges and the possible alternatives are presented in more details hereafter.

A. Clock Drifting

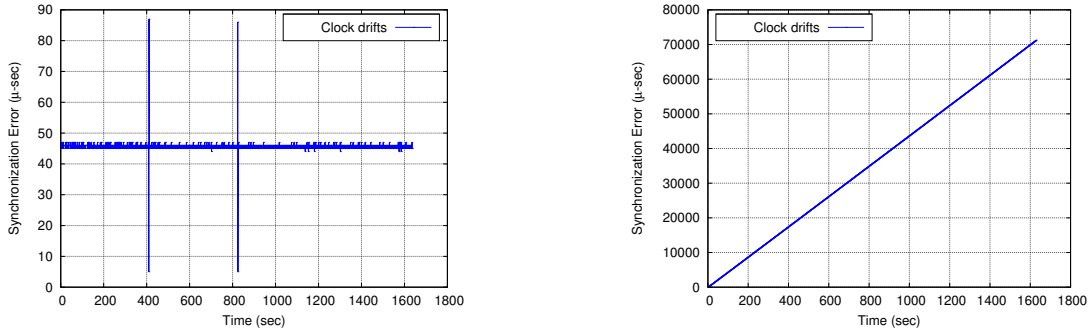
Cost reduction of the mote circuitry and components inevitable leads to the use of cheap but less reliable components, such as clock crystals. A reliable crystal ticks at a very stable frequency, which enables to precisely measure the time, where low-cost crystals deviate from their theoretical frequency over time, which is known as clock drifting. The fast drifting is the most impacting feature of clocks used in today's sensor motes. To investigate the clock drift, we performed an extensive experimental study with MICAz motes, one of the largely used platforms. Relative drift between two motes have been investigated. Motes have been wired through the available *pins* with an Arduino that periodically and simultaneously submits a signal to the motes, which has been used to capture the instantaneous clock values. Log files have been used to calculate the instantaneous clock differences and the cumulative ones, i.e. the differences due to the drift between two consecutive clock values, and those due to the drift cumulated from the beginning of the experiment. Results show drifting in the order of tens of microseconds (Fig. 1 (a)) with $50\mu sec$ as a baseline and continuous rise and drop of the order of few microseconds, with some exceptions of picks at the order of more than $90\mu sec$ rise followed by a decrease at less than $10\mu sec$. Several executions with different nodes show similar forms as the one reported here, with possible minor difference in the base-line and pick values. This results in almost a *linear* cumulative increase, refer to Fig. 1 (b). In addition to the drifting due to the inherent features of the crystal, dynamic environmental conditions such has the ambient temperature have also an impact. Investigating such issues is beyond the purpose of this paper, but it worthy to point out here the work by Yang et al. [4] that proposes techniques to compensate for clock drift caused by such environment dynamics, which are practical and can be integrated in future synchronization protocols.

The clock drifting makes estimated offset out-of-date over time, notably if a high precision synchronization is required. Re-synchronizing the nodes frequently– if needed– may be very costly in terms of communication overhead and energy consumption. The alternative is to capture the rate of the drift (skew) in the estimation model, which is of high importance for long-term synchronization. The results reported in Fig 1 (b) confirm suitability of the linear model for the skew estimation. To quantitatively investigate the impact of skew estimation on long-term synchronization, we conducted an experiment where synchronization error between two MICAz motes has been measured in both models (offset-only and skew-offset) and using the basic sender-to-receiver synchronization. Synchronization messages have only been exchanged for few rounds to permit nodes estimate the offset (and skew in the joint model), then the synchronization process has been stopped and the synchronization error has been measured, i.e. $t = 0$ in the plot is the time at which the synchronization message exchange are stopped. An Arduino connected to synchronizing motes via available *pins* has been used to generate signals simultaneously at both nodes, which enabled to make simultaneous measurements. Results that are represented in Fig. 2 show large difference between the offset-only model and the skew-offset model. The error for the latter remains below $10\mu sec$ for several tens of seconds, and at the microsecond level (below $300\mu sec$ during the whole $15min$ experimentation), where it increases sharply in the offset-only model and reaches the millisecond level after few seconds. This confirms that offset-only solutions are not useful for longtime synchronization. Despite its instability, the offset-only model has the advantage of simplifying estimator calculation. This makes the offset-only model appropriate for applications that need sporadic delay-tolerant synchronization, where the synchronization can be performed instantly before the timestamping of the reported event. This would reduce memory footprint compared to the skew model without a significant impact on the communication overhead and the energy consumption.

B. Mote Limitations

Most simulations and mathematical analysis use tools such as Matlab or C-based simulators at a high level of abstraction. They consist in coding for a standard computer (PC), running and collecting the results for analysis of thorough, but computation-costly estimators. However, they completely ignore that this code for calculating estimators is to be run by memory and computation limited sensor motes. For example, the floating-point computation is not supported by most platforms (MICAz, TelosB, etc.); but it may be implemented as a library at the kernel of operating systems, e.g, TinyOS2 [1]. Our experience with TinyOS revealed several problems with this library when handling large numbers (clock values), and re-implementation of floating-point division calculation has been necessary. Estimators evolving sequences of clock values multiplications are to be avoided, as they cause fast

[1] http://www.tinyos.net/



(a) (b)

Fig. 1: Clock Difference of Internal Clocks a) Instantaneous, b) Cumulative

arithmetic overflow due to the large size of the clock values and the limited variables holding such values. Such estimators should be simplified and/or rewritten to fit motes limitations. Most estimators rely on the collection of large samples to get statistical meaningful estimates. This would have an important memory footprint on the sensor motes; Techniques such as the use of limited window with possible moving average can be helpful. However, the results would be different from the nice theoretical performance; an issue that needs to be addressed by testbed evaluations. Existing solution such as *TinyECC* [2] in TinyOS and its NN library can be used to implement estimators to overcoming the problem of arithmetic overflow, as they enable customizing the variable size according to the user need. However, the memory space should be carefully managed since the memory footprint may increase dramatically.

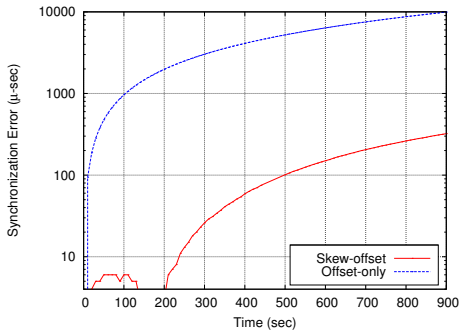


Fig. 2: Synchronization error vs. time

C. Delay Variation

All synchronization protocols rely on some sort of message/signal exchange between nodes. The time involved in sending a message is the result of the following four factors that can vary non-deterministically [2]: i) Send time, which is the time spent by the sender for message construction added to the time spent to transmit the message from the sender's host to the network interface. ii) Access time; the time spent for medium access at the MAC layer. iii) Propagation time, as the time for the message to reach the receiver once it has left the sender's radio. iv) Receive time, which is the time spent by the receiver to process the message. The sender-to-receiver involves the four factors, while the receiver-to-receiver approach removes the send time and the access time from the critical path [3, P. 294], which represents the two largest sources of non-determinism. This reduces the effect of delay variation. Most estimation methods assume delays to follow certain distributions (Gaussian and exponential distributions are the most used), and estimators are derived accordingly. Other practical approaches consider to use low layer timestamping (MAC layer), as well as mechanisms like timestamping several bytes to normalize the jitter caused by coding and decoding [5]. This considerably reduces the amplitude of the variation, which allows to achieve relatively high degree of precision. However, it makes the implementation dependent on the specific platform and reduces the portability compared to high layer timestamping. The big challenge caused by delay variation is when using timestamping at the higher layers, which would be at the order of several tens and even hundreds of microseconds. High layer timestamping has the advantage of large portability.

IV. Impact of Empirical Parameters

Empirical parameters have a significant impact on the protocol performance, which has been ignored in the theoretical studies. This important issue is investigated in this section. The elementary sender-to-receiver approach for single-hop synchronization and the joint skew-offset model is considered. Similar experimentation set-up and estimators as in Sec. III-A have been used, i.e. two MICAz motes with Arduino for instantaneous clock value capturing, and MLE for offset/skew estimation in Gaussian (normal) model [1]. Note that the two nodes exchange a two-way message to construct timestamps that form a sample. The process is then repeated for a certain rounds to acquire a set of samples for estimation. We fixed the sample size, say k, to 15 and investigate the impact of the

[2]http://discovery.csc.ncsu.edu/software/TinyECC

period separating two rounds, say T, on the synchronization error. Two values of the latter have been investigated, $50msec$, and $100msec$. Note that $t = 0$ is the time when the estimation is completed and the protocol execution is stopped (similarly to III-A).

The results presented in Fig 3 show that the parameter T has an impact on the synchronization error, and the lower value (dashed lines) provides better performance than the higher one (solid lines). ML estimators rely on the assumption that the transmission/reception delays follow a normal distribution with the same parameters, i.e., $\mathcal{N}(\mu, \sigma^2)$. Theoretical analyses confirm that the precision of the estimators and their lower-bound, $CRLB$, inversely depends on the variance, σ^2, but no study investigated the issues that may affect σ^2. We remarked in the experiment that the variance of the measured delays for executions with $T = 50$ were lower than those with $T = 100$. This may be justified by the fact that channel conditions vary over time, and that picking up samples in shorter periods is likely to encounter more similar conditions than picking them up throughout longer periods, which fulfills the assumption with a lower variance (σ^2) and consequently permits to achieve better estimators.

It is thus strongly recommended to perform sample acquisition in one cycle (round) when using models relying on assumptions about the delay distribution (such as MLE), and the skew/offset model, notably in low duty-cycled applications. The duty-cycle may be established using a cycle with a synchronization phase(s) that should be long enough to fit operations needed to get the sample size k, followed by a set of cycles without a synchronization phase, rather than using a synchronization phase at the beginning of every cycle.

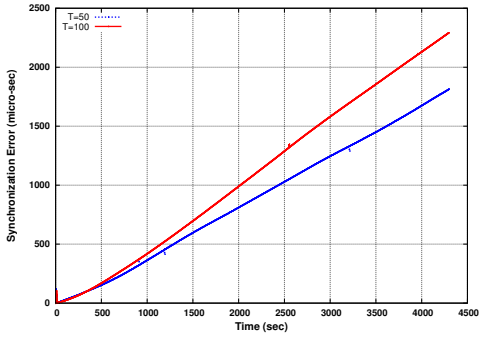


Fig. 3: Impact of Empirical Parameters

V. Synchronization Protocols

Some synchronization protocols providing microsecond level precision are described below. Descriptions and discussions in this article are focusing on implementation issues. For detailed presentation of each protocol, the reader can refer to other surveys or the original articles introducing the protocols.

A. Reference Broadcast Synchronization (RBS)

RBS [6] is the first protocol that introduces the receiver-to-receiver concept to WSN. The authors measured reception delays via extensive experiments, where resulted samples fitted Gaussian distribution. This result has been used to derive estimators, and it conducted estimator calculation methods for most receiver-to-receiver protocols proposed later. Both offset-only and joint skew-offset estimation models have been considered, where simple linear regression has been used to estimate the skew from the offsets (motivated by the confirmed zero mean Gaussian distribution of the delay differences). Further, the protocol supports both high layer timestamping and low-layer timestamping.

Berkeley Mote has been used, running TinyOS. RBS has also been implemented and tested in a commodity hardware platform with Lucent Technologies, $11Mbit$ 802.11 wireless Ethernet adapters, running UNIX daemon with UDP datagrams. This implementation has been carried out for testing low-layer timestamping that was not been possible with TinyOS and the Berkely mote of that epoch. Six motes have been used in the experiments, with one as a reference. Results for high-layer timestamping show error at the order of few micro seconds, where more than 99% of errors have been below $30\mu sec$ for light traffic, and $49\mu sec$ for heavy traffic. Unix kernel implementation with low-layer timestamping implementation demonstrated errors at the order of $1\mu sec$ to $6\mu sec$, with an average below $2\mu sec$. Offset conversion in multi-hop environment has also been proposed and tested in a linear topology with 5 nodes and five references, where results demonstrated smooth increase of the error that did not exceed $11\mu sec$ for four hops (with low-layer timestamping). These results clearly demonstrate the effectiveness of the receiver-receiver approach. A notable feature of RBS implementation is that both timestamping schemes have been evaluated, which confirms that the high level of abstraction in the conception with regard to timestamping does not eliminate the possibility of low-layer timestamping, which enables high precision but with reduced portability. This aspect has been ignored by many protocols proposed ahead of RBS, which wrongly assume RBS only supports high-layer timestamping, e.g. TPSN [7] that is presented next.

B. Timing-Sync Protocol for Sensor Networks (TPSN)

TPSN [7] is one of the earliest synchronization protocols proposed for WSN, which is built upon a tree-topology where all nodes are synchronized to the root (sink). Since it uses global synchronization where all nodes are synchronized to the sink on a tree-topology, it is more appropriate to periodic applications. Nonetheless, maintaining such a tree and global synchronization is not justified in event-based applications and would be energy inefficient. Synchronization is achieved by gradually running the sender-to-receiver approach on the constructed tree. MICA motes have been used in the implementation along with TinyOS. RBS has also been implemented for comparison. The maximum frequency of the crystal used in motes is $4Mhz$, which permits to achieve a granularity of $0.25\mu sec$. Results show synchronization error of few microseconds, not exceeding $45\mu sec$ in the worst case for single-hop scenarios, and $74\mu sec$ for 5-hop scenarios. The results

show that in most cases, the error has almost been halved compared to RBS. However, the authors implemented TPSN with MAC layer timestamping, but they use application layer timestamping for RBS. This is an unfair comparison and leads to questionable conclusions. RBS concept is independent from the timestamping level, as it supports both high-layer and low-layer timestamping. Further, RBS low-layer timestamping has been implemented and tested, [6, p. 157]. A fair comparison would use an implementation of the same timestamping layer, in which case RBS would be more accurate due to the receiver-to-receiver feature (Sec. III-C).

C. Flooding Time Synchronization Protocol (FTSP)

Maroti et al. [5] proposes FTSP, a tree-topology synchronization protocol where all nodes synchronize to the root node (sink). The protocol is the first that considers interrupt jitter compensation by recording several timestamps per packet both at the sender and receiver sides, then averaging and normalizing the obtained timestamps to come out with a final timestamp of the outgoing message. Tests with MICA2 motes timed with a $7.37MHz$ clock and running TinyOS demonstrate microsecond-level precision on average in a topology of 60 nodes. Linear regression has been used for the skew compensation as in RBS. A thorough investigation on the effectiveness of the skew estimation (long term synchronization) has been presented, where synchronization have been stopped once nodes get synchronized and then error are measured, similarly to the experiment presented in Sec. III-A. Results show microsecond level precision for several minutes. Nonetheless, no comparison with the offset-only model has been provided. A similar investigation on the latter model would be useful to clarify the benefit from the skew estimation.

The high precision of FTSP implementation is basically due to the effective jitter compensation technique. FTSP implementation is available in the official distribution of TinyOS. The first impressive feature of the implementation was its portability with all platforms including those with packet-oriented radios such as $CC2420$, e.g. MICAz and TelosB, which do not enable byte-oriented interrupt triggering. However, after careful analysis of the code we realized that the external clock has been used, with millisecond level interfaces, and that only one timestamping per packet is used instead of the proposed multiple timestamping. The current implementation is thus far from the high precision reported in the original paper. For protocol comparison with FTSP at the microsecond level, a careful modification of FTSP implementation is then required.

D. Gradient Time Synchronization Protocol (GTSP)

Summer et al. [8] focus on the local synchronization (between direct neighbors) for which high precision is targeted. Similarly to FTSP, GTSP uses several timestamping per packet for jitter compensation. But contrary to FTSP, the protocol does not require a tree topology and does not use the wall-clock model, but the distributed logical clock concept. Nodes periodically broadcast synchronization packets and logical skew and offset are calculated at each node using simple averaging. A formal proof of convergence is provided. GTSP has been implemented in MICA2 with TinyOS. $Timer3$ of ATmega128L has been used that operates at $1/8$ of the external oscillator frequency, i.e., at $921kHz$. The authors argue that packet-oriented radio chips like $CC2420$ (MICAz, TelosB) do not allow compensation of jitter in the interrupt handling time since they trigger an interrupt after the whole packet reception instead of doing it byte-by-byte. This– added to the relatively high precision of its external clock– justifies the use of MICA2. GTSP has been compared with FTSP using 20 nodes in a ring logical topology. Results show slightly better performance in favor of GTSP for local synchronization (an average of $4\mu sec$ vs. $5.3\mu sec$), but slightly less performance for multi-hop synchronization (an average of $14\mu sec$ vs. $7\mu sec$). This gradient property is very useful for applications where multi-hop synchronization is of less importance.

TABLE I

Protocol	Scope	Topology	Drift estimation	Platform
TPSN	Multi-hop	Tree	No	MICA+TinyOS
FTSP	Multi-hop	Tree	Yes	MICA/MICA2+ TinyOS
RBS	Multi-hop	Flat	Yes	Berkeley mote+TinyOS
GTSP	Multi-hop	Flat	Yes	ICA2+ TinyOS
Glossy	Multi-hop	Tree	No	TmoteSky+Contiki
R^{4}Syn	Single-hop	Flat	Yes	MICAz TinyOS

E. Glossy

Glossy [9] combines network message flooding and synchronization in a unique protocol. It exploits constructive interference of IEEE $802.15.4$ packets, and targets network wide synchronization. Contiki operating system and Tmote sky sensor motes have been used in the implementation. The latter integrates $MSP430$ microcontroller and $CC24220$ radio. Two clocks have been used; the high-frequency DCO and the low-frequency external crystal. Similarly to HARMONIA [10] [3], the virtual high-resolution time (VHT) technique has been employed to translate the high-resolution estimate of the reference time to a low-resolution value with a high-precision at the external crystal clock. The latter is then used to implement duty-cycle and coordinate flooding. These two time values are provided to the application. Influenced by FTSP, Glossy compensates software jitter by measuring the gap between interrupt reception and interrupt service, then accordingly inserting a certain number of non-operations (NOP) at the beginning of the interrupt handler. Synchronization error has been evaluated in a controlled setting using a couple of nodes, while message flooding of Glossy has been evaluated in an extensive experiment on testbeds. Results of the controlled experiments have showed an average error below $1\mu sec$, even for multi-hop synchronization.

[3] note that this protocol is not presented in this paper as its implementation does not demonstrate high precision

F. R^4Syn

R^4Syn [11] is a distributed receiver-to-receiver protocol based on RBS, where all nodes cooperatively assure the traditional role of the reference in a round-robin way. Beacons and timestamp exchange are integrated in the same steps to accelerate sample acquisition and estimator calculation. Djenouri et al. [12] provide a fault-tolerant implementation of R^4Syn, which tolerates temporary and permanent failure of nodes without affecting correct behavior of the protocol. In the experiment, three MICAz motes have been used with TinyOS. An Arduino has been used for measuring the synchronization error. Two motes whose synchronization error is to be measured have been connected via available *pins* to the Arduino that periodically triggers measurement. R^4Syn allows both low-layer and high-layer timestamping (like RBS), but only low-layer timestamping has been evaluated. Both skew-offset and offset-only models have been implemented. Extensive tests show errors below $10\mu sec$ in most cases, with 95% confidence interval. Original MLE estimators that results in multiplications of very large temporary variables before convergence have been analytically rewritten to avoid such problem, and they have been successfully implemented.

A comparison between the offset-only and skew-offset models has been carried out, similarly to the experiment presented in Sec. III-A. This gives a clear picture on the drift consideration impact. However, a major lack in the experiment is with regard to multi-hop evaluation, and the limited scope of the evaluated network (only three nodes). Further, contrary to RBS, no investigation on high-layer timestamping has been provided, which is an important feature of R^4Syn. Table I Summarizes the features of protocol implementations presented in this paper.

VI. Conclusion

Implementation of synchronization protocols in wireless sensor networks (WSN) has been investigated in this article. Such implementation faces many challenges, and many aspects have been neglected in the theoretical solutions. First and foremost, the clocks used in WSN suffer from the high drifting, which limits the validity of the offset estimators over time. The drifting of MICAz has been quantitatively investigated in this paper, where the results show fast drifting of its clock. Frequent re-synchronization is energy inefficient, and skew estimation models represent the vital solution to this problem. However, the implementation of skew models comes at a higher computation complexity compared to the offset-only models, and therefore higher memory footprint. Our results confirm suitability of linear models for skew estimation.

For real implementations, estimators should be simplified, e.g., the use of approximated forms such as limited window moving average may be helpful. Offset-only implementations do not permit long-term synchronization. They can only be used in application requiring sporadic delay-tolerant synchronization, where the synchronization can be performed instantly before the timestamping of the reported event. The offset-only model can also be useful for applications that may be satisfied with a weak synchronization. Another big challenge is the delay of message exchange/handling, which is prone to high variability. MAC layer timestamping is a practical solution, but comes at a reduced portability. Estimation models that capture delay variability allow high-layer platform-independent implementation.

One of the practical approaches in real implementation is to trading-off the synchronization precision for a reduced cost. The precision should be bounded according to the application needs. Another practical technical is the use of several timestampings for every synchronization message and averaging the resulted timestamps to compensate the jitter, e.g. FTSP, GTPS. But this comes at a reduced portability and can only be implemented with byte-level interrupt triggering radios, e.g. MICA2, and not the common packet oriented radios, e.g. MICAz and TelosB. This makes protocols like FTSP and GTPS more suitable for the platforms with the former radios. Many protocols (e.g. TPSN) have been compared with the TinyOS distribution of FTSP that uses millisecond interfaces and does not implement the multiple timestamping per packet, which is FTSP's main contribution. For precision investigation, a more fair comparison would use byte-level interrupt triggering platforms and a revisited implementation.

References

[1] Y.-C. Wu, Q. M. Chaudhari, and E. Serpedin, "Clock synchronization of wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 28, no. 1, pp. 124–138, 2011.

[2] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: A survey," *IEEE Network Magazin*, vol. 18, no. 4, pp. 45–50, 2004.

[3] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad hoc Networks*, vol. 3, no. 3, pp. 281–323, 2005.

[4] Z. Yang, L. Cai, Y. Liu, and J. Pan, "Environment-aware clock skew estimation and synchronization for wireless sensor networks," in *IEEE INFOCOM*, 2012, pp. 1017–1025.

[5] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *SenSys*, 2004, pp. 39–49.

[6] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *5th USENIX Symposium on Operating System Design and Implementation (OSDI'02)*, December 2002.

[7] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ser. SenSys '03, 2003, pp. 138–149.

[8] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *Proceedings of the 8th ACM International Conference on Information Processing in Sensor Networks (IPSN'08)*, 2009, pp. 37–48.

[9] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *IPSN*, 2011, pp. 73–84.

[10] J. Koo, R. K. Panta, S. Bagchi, and L. A. Montestruque, "A tale of two synchronizing clocks," in *Proceedings of the 7th ACM International Conference on Embedded Networked Sensor Systems (SenSys'09)*, November 2009, pp. 239–252.

[11] D. Djenouri, "R^4syn : Relative referenceless receiver/receiver time synchronization in wireless sensor networks," *IEEE Signal Process. Lett.*, vol. 19, no. 4, pp. 175–178, 2012.

[12] D. Djenouri, N. Merabtine, F. Z. Mekahlia, and M. Doudou, "Fast distributed multi-hop relative time synchronization protocol and estimators for wireless sensor networks," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2329–2344, 2013.