Synchronization Protocols and Implementation Issues in Wireless Sensor Networks: A Review

Djamel Djenouri, Miloud Bagaa

***Abstract*—Time synchronization in wireless sensor networks (WSN) is a topic that has been attracting the research community in the last decade. Most performance evaluations of the proposed solutions have been limited to theoretical analysis and simulation. They consequently ignored several practical aspects, e.g. packet handling jitters, clock drifting, packet loss, mote limitations etc, which effect real implementation on sensor motes. 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 paper gives an insight 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 testbed evaluation are motivated by some experiments we conducted. The most relevant implementations of the literature are then reviewed, discussed, and qualitatively compared. While there are several survey papers that present and compare the protocols from the conception perspectives, and others that deal with mathematical and signal processing issues of the estimators, a survey on practical aspects related to the implementation is missing. To our knowledge, this paper is the first one that takes into account the practical aspect of existing solutions.**

I. Introduction and Background

A. Introduction

Time synchronization is of high importance for many applications and protocols in wireless sensor networks (WSN). For instance, in a moving object (e.g. vehicle) tracking application, sensor nodes report the location and time at which they detect the object to a base-station, which combines the obtained information to estimate the location and velocity of the tracked object. Nodes should be synchronized to correlate the different reports. Another example is in duty-cycling and contention-based channel access scheduling, where nodes coordinately switch between active and sleep modes. Time synchronization is also required for many other applications in WSN, such as data fusion/aggregation, TDMA scheduling, realtime monitoring and actuation, etc. Time synchronization has always been one of the fundamental and challenging problems in distributed systems. The lack of a shared memory makes exchange of high-layer messages or low-layer signals between nodes inevitable for protocol construction. The high delay variability of communications in WSN, added to node limitations (computation, memory, energy), elevate the complexity of the problem.

Several protocols for time synchronization in WSN have been proposed in the literature. The evaluation of the proposed solutions can be divided into two categories; i) analysis and simulation-based evaluation, vs. ii) empirical evaluation. The first category includes the use of network simulations for comparison with state-of-the-art candidates. It also includes numerical analysis of estimators and possible comparison of the mean square errors (MSE), or its variants, with an optimum, e.g. Cramer-Rao lower-bound (CRLB) [1]. This provides a preliminary vision on the protocol performance, and it is essential for investigating issues that are difficult to evaluate with real tests, such as scalability. Nonetheless, it cannot replace 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 where the protocols are implemented on real motes and evaluated in a testbed experiment is thus vital to get a concrete view on the synchronization protocol, its features and limitations. Existing implementations can be reused in other applications or by other protocols. Therefore, having a horizontal vision on current implementations is essential to decide which implementation can be reused adequately to fulfil one application's requirements or another, or which one can be adapted with minimum amendments. The aim of this paper is to throw some light on issues related to the implementation of a synchronization protocol, to present and discuss state-of-the-art implementations.

The rest of the paper is organized as follows. The remainder of this section introduces some general concepts that are used throughout the paper. The related work is summarized in the next section, followed by the implementation challenges with some experimental illustrations in Section III. Section IV presents our investigation on the impact of some empirical parameters. Implementations of the literature are reviewed Section V. Section VI provides discussions and summarizes the lessons we have learnt. Finally, Section VII concludes the paper.

B. General Concepts

1) Skew/Offset vs. Offset-Only: A hardware oscillator is used to implement the sensor mote's clock (similarly to any computing device). Let $C(t)$ denotes the value of the clock at time t. The oscillator frequency determines the rate at which the clock runs. The rate of an ideal clock would equal one, i.e. its derivative vs. time ($\partial C/\partial t = 1$). However, in practice the clock frequency varies unpredictably due to

The authors are with CERIST Research Center, Algiers, Algeria. Email: ddjenouri@acm.org, bagaa@mail.cerist.dz

various physical effects, e.g, temperature, crystal aging, etc. This is known as clock drifting. The clock value is then approximated using appropriate estimators. Two estimation models can be distinguished, the offset-only model vs. joint skew/offset model. In the first model, the local clock at some node, say n_i, is approximated by [2]:

$$C_i(t) = t + \theta_i, \tag{1}$$

where, θ_i, is the offset of node i's clock to an ideal clock (realtime). Relative equation relating two nodes' clocks, C_i and C_j, can be given by:

$$C_j(t) = C_i(t) + \theta_{n_i \to n_j}, \tag{2}$$

where, $\theta_{n_i \to n_j}$, represents the *relative* offset relating time at node, n_i, to the corresponding one at node, n_j.

This model has the simplicity advantage, but does not ensure long-term estimation as it does not capture clock drifting. Therefore, synchronization messages need to be exchanged at a high frequency to assure good precision. The second model provides long-term estimators at the price of augmented calculation complexity. The local clock of node, n_i, can be approximated by [3],

$$C_i(t) = \alpha_i t + \beta_i, \tag{3}$$

where α_i is the absolute *skew*, and β_i is the *offset* of node i's clock. Relative equation relating two nodes' clocks, C_i and C_j, is given by,

$$C_j(t) = \alpha_{n_i \to n_j} C_i(t) + \beta_{n_i \to n_j}, \tag{4}$$

where $\alpha_{n_i \to n_j}$ and $\beta_{n_i \to n_j}$ represents the *relative* offset and skew, respectively.

All synchronization protocols proposed for WSN use some sort of estimation for synchronization parameters, and they thus fall either into the offset-only class or the joint skew/offset class.

2) Sender-to-receiver vs. Receiver-to-receiver: In the sender-to-receiver model, nodes periodically exchange timestamped synchronization messages. They synchronize to one another using the timestamps of submission and reception events [4]. The receiver-to-receiver approach exploits the property of the physical broadcast medium. A common reference is used for broadcasting synchronization messages, and then receivers within the reference's vicinity exchange the time at which they receive the same message to get synchronized. Fig. 1 illustrates the construction of one sample in the two approaches. In the sender-to-receiver approach, the transmitter (node n_1) timestamps the first packet as t_1, the reception of this packet at n_2 is timestamped by the latter, t_2, and then the transmission/reception of the reply packet are respectively timestamped t_3, t_4. The tuple (t_1, t_2, t_3, t_4) is used as a sample for estimation. The process is repeated for k rounds to construct k samples, then the parameters are estimated. In the receiver-to-receiver approach, a reference is used to broadcast messages, and only reception timestamps are used to construct samples (t'_1, t'_2). As show in Fig 1, the receiver-to-receiver approach has the advantage of reducing the time-critical path,

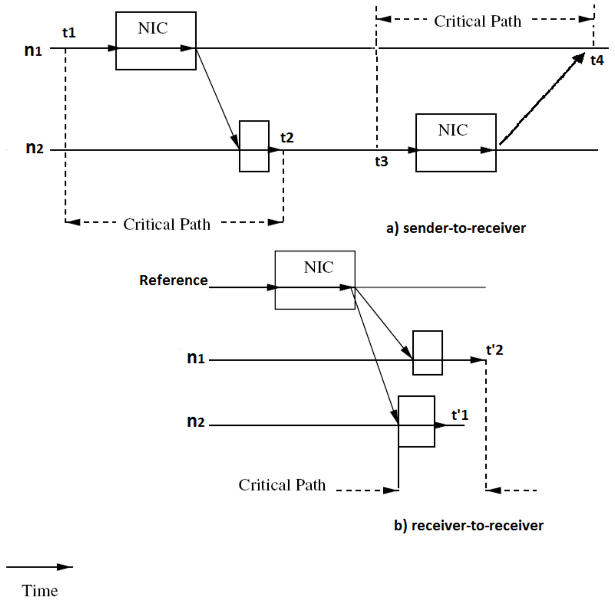

Fig. 1: Receiver-to-receiver vs. Sender-to-receiver

and therefore improving synchronization accuracy compared to the sender-to-receiver approach [4]. The time-critical path is the latency that contributes to non-deterministic errors when exchanging synchronization signals. The time-critical path of the first approach is the result of the following four factors that can vary non-deterministically [3]: i) Send time; spent by the sender for message construction and transmission to the network interface, ii) medium access time (at the MAC layer), iii) propagation time, and iv) receive time at the receiver to process the message. The receiver-to-receiver approach removes the send time and the access time from the critical path [4].

II. Related Work

Several survey papers on time synchronization in WSN have been published in the last few years. In [3] by Sivrikaya and Yener, three categories of synchronization have been reported, 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 belong to this category. Meanwhile, synchronization to a reference model also called the "always on" model, or global synchronization, is the most complex, as it requires all nodes maintain their clocks synchronized to a single reference in the network. The goal of this type of synchronization is to preserve a global timescale throughout the network.

Sundararaman et al. [4] 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. peer-to-peer, ii) clock correction vs. clock untethered, iii) internal vs. external, iv) probabilistic vs. deterministic, and v) sender-to-receiver vs. rec-to-rec. In the master slave model, all nodes (slaves) synchronize to a single master, while nodes are pair-wise synchronized in the peer-to-peer model without using a single reference. This eliminates the single point of failure. The clock untethered model (relative synchronization) allows nodes to run their clocks independently without the need for continuous update of the clock variable, contrary to the clock correction model. External synchronization refers to an external source from the network for a standard source of time (such as universal time). The deterministic and probabilistic aspects are with respect to the offset bound guarantee on the synchronization. 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 synchronization messages.

Another interesting recent survey paper 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. Solutions and estimators are analyzed with respect to Gaussian, exponential, and arbitrary distributions for message exchange delays. The authors show that the optimal estimators are relatively easy to derive for Gausian delays, where the minimum variance unbiased estimator(MVUE), best linear unbiased estimator (BLUE), maximum likelihood estimator (MLE), and least square (LS) estimator all coincide. For Exponential delays, the authors analyze estimators based on MLE, best linear unbiased estimation using order statistics, MVUE with Rao-Blackwell-Lehmann-Scheffe theorem. For the sender-to-receiver approach, they analytically demonstrate that MLE (the most largely used in the literature) is better than the MVUE when the means of the up-link and down-link delays are very close to each other, and that the MVUE becomes better when the up-link and down-link delays are dispersing. The lack of estimators for the receiver-to-receiver protocols in environments with exponential delays is reported, which represent an open research trend. For arbitrary delays, some estimators based on linear programming (LP), boot strap bias correction, composite particle filtering are presented. These estimators are robust when delay distributions are unknown, and they can adapt to different delay distributions. It is reported that the MSE performance of bootstrap bias corrected estimate is better than the exponential MLE when applied to non-exponential delays. To illustrate this, the authors analytically compare the performance of the MLE of clock offset derived under exponential delay and its corresponding boot strap bias corrected estimator, when applied to Gamma distributed delays with two degrees of freedom. As a perspective, the authors discuss the application of distributed signal processing techniques (e.g.,distributed estimation and detection), which should be helpful to derive distributed clock synchronization algorithms with optimal ways for information passing. This will save unnecessary communication overhead. Finally, the potential benefit from jointly solving the localization and synchronization problems (that are strongly related) is mentioned [5], where estimators such as M-estimator becomes relevant.

A survey similar to [1] from the empirical and practical perspective is missing in the current literature. This represents the aim of the paper, where the real implementations of synchronization 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, added to mote limitations are inevitable consequences that result from reducing the sensor mote cost and size for economic reasons. 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 environment, to which add long jitters (delay variation) for in-node message processing at sensor nodes that is caused by the mote limitation and instability. The faulty-nature of motes and lossy channels are other challenges that faces an implementation of a synchronization protocol, which make fault-tolerance a must before real deployment. Finally, accuracy measurement needs some hardware manipulation and complicates large scale experimentation. All these challenges and the possible alternatives are presented in more details hereafter.

A. *Fast Drifting*

One of the most influencing features of clocks used in today's sensor motes is the fast drifting. This is due to the circuitry cost reduction that inevitable requires the use of cheap, but less reliable components. Most motes include two clocks: The first is the internal clock that allows for microsecond level granularity, but also with relatively a high drifting, as it will be illustrated later. The second type is the external crystal clock, which is more stable but usually has low frequency and thus provides a weak granularity. For instance, crystals used in MICAz and TelosB have $32.768KHz$ frequency, i.e. their granularity bound is $30.5\mu sec$. The external clock has the advantage of running when the node turns to the sleep mode, contrary to the internal clock. This justifies the low frequency use for the sake of power saving. To investigate the clock drift, we performed an extensive experimental study with MICAz motes, one of the largely used platforms. First, relative drift between two motes has been investigated. We have wired two motes through the available *pins* with an Arduino that periodically and simultaneously submits a signal to the motes, to capture the instantaneous clock values (Fig. 2). We have then used log files to calculate the instantaneous clock differences and the cumulative ones, for both the internal clock, Fig. 3, and the external one, Fig. 4. Results show few tens of instantaneous tick drift for the internal clocks, Fig. 3 (bottom).This is with 50 ticks as a base-line and continuous successive rise and drop of the order of few ticks, with some exceptions of picks at the order of more than 90 ticks rise followed by a decrease at less than 10 ticks. Several executions with different nodes showed similar forms as the one reported here, with possible minor differences in the base-lines and pick values. This results in almost a linear cumulative increase. Linear cumulative increase is also noted for the external clock, Fig. 4. However, the latter shows more stability where the instantaneous drift alternates between 0 and 1 tick.

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. The alternative for ensuring long-term synchronization is to capture the rate of drifting (skew) in the estimation model (Eq. 4). The results reported here 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 protocol approaches (send-to-receiver and receiver-to-receiver). This is using the internal high precision clocks.

For each model and synchronization paradigm, synchronization messages have only been exchanged for few rounds to permit nodes estimate the offset (and skew in the joint model). The synchronization process has then been stopped and the synchronization error has been measured, i.e. $t = 0$ is the time at which we stopped the synchronization message exchange. As the synchronization messages were not queued (no queuing delays), the standard MLE of the Gaussian distribution delay has- been used in both models [1]. We used an Arduino to generate simultaneous signals at both nodes, similarly to the previous experiment. The results for the sender-to-receiver synchronization are illustrated in Fig. 5. Similar results have been obtained for the receiver-to-receiver synchronization, and thus the figure of the latter is omitted for space limitation. Fig. 5 shows a large difference between the offset-only and the skew-offset. The skew-offset estimation keeps the error below $10\mu sec$ for several tens of seconds, and at the microsecond level (below $300\mu sec$ during the whole $15min$ experimentation). However, in the offset-only model, the error reaches the millisecond level after few seconds.

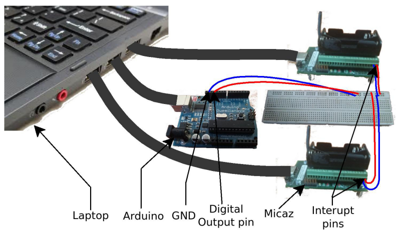

Fig. 2: Experimentation Setup

B. Mote Limitations

Most simulations and mathematical analyses use tools such as Matlab or C-based simulators at a high level of abstraction. They consist in coding for a standard computer, running and collecting the results for analysis of thorough,

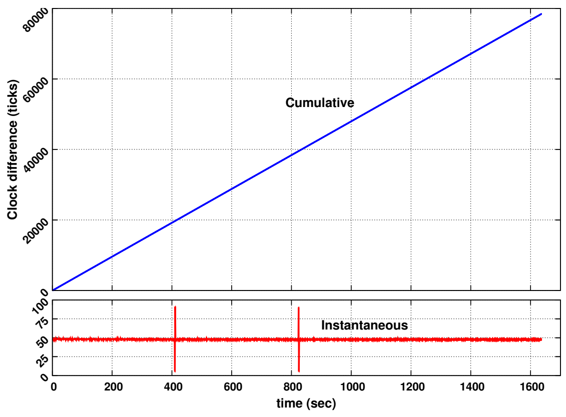

Fig. 3: Clock Difference of Internal Clocks

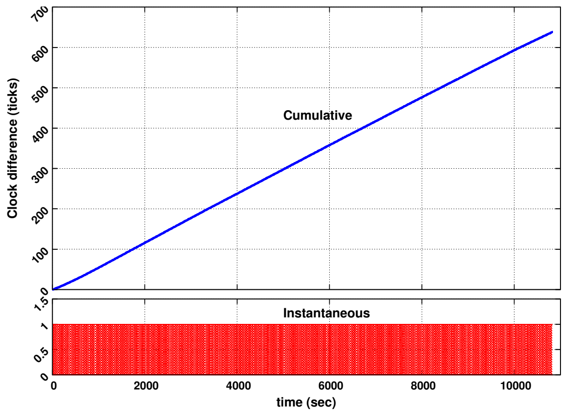

Fig. 4: Clock Difference of External Clock

but computation-costly estimators. However, they completely ignore that this code for calculating estimators is to be run by sensor motes, which are limited in memory and computation capacity. 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 arithmetic overflow due to the large size of the clock values and the limited size of their respective variables. 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. The use of limited window with possible moving average can help reducing the memory footprint. However, the results would be different from the nice theoretical performance; an issue that needs to be addressed by testbed evaluations. Existing solutions such as *TinyECC* [2] of TinyOS, and its NN library can be used to implement estimators while overcoming the

[1]http://www.tinyos.net/
[2]http://discovery.csc.ncsu.edu/software/TinyECC

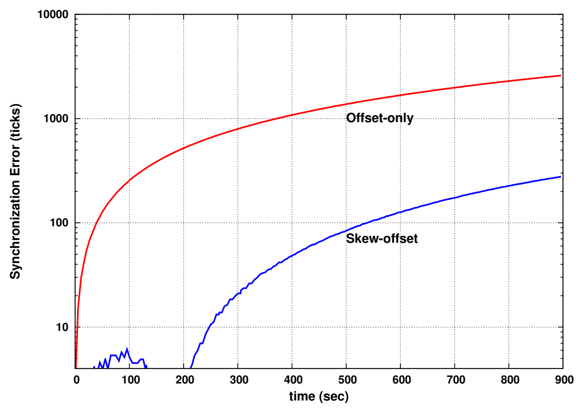

Fig. 5: Synchronization Error of the Skew-offset vs Offset-only Model

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.

C. Delay Variation

All synchronization protocols rely on some sort of message exchange between nodes. As explained in Sec. I-B, the receiver-to-receiver approach removes the send time and the access time from the critical path, 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, then estimators are derived accordingly. Other practical approaches consider low layer timestamping (MAC layer), as well as mechanisms like timestamping several bytes to normalize the jitter for coding and decoding [6]. 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, where the delays would be at the order of several tens and even hundreds of microseconds, but this has the advantage of large portability [4].

D. Fault Tolerance and Security

Most theoretical and simulation-based evaluation suppose ideal scenarios and neglect aspects such as packet loss, and node temporary or permanent failure. Real implementation provides more accurate vision on the performance in real scenarios. But first, one should ensure the correct behavior of the implemented protocol in the presence of faulty nodes. For example, protocols using consensus for time update (such as averaging offsets [7]) are affected by erroneous reports. More importantly, those based on round-robin operation (e.g. [8], [9]) may fail and stop working when a single node is down. Many solutions make abstraction of this aspect in the protocol design. Indications like the absence of messages or responses from a node during a threshold time may be used as an indicator of faulty nodes[10], but the threshold should be carefully selected to avoid possible false positives due to the lossy channels. In addition to faulty nodes, considering Byzantine behavior is challenging [11], where compromised nodes may report falsified timestamp values for attacking the synchronizing protocol.

E. Accuracy Measurement

To measure the accuracy of a synchronization protocol (the synchronization error), instantaneous local times (or estimates) at the synchronized nodes involved in the measurement is needed to be captured red at the *same physical time*. This is similar to the fundamental atomic snapshot in distributed systems. However, existing distributed solutions are not useful in this case due to the high variability of message exchange. The trend in most measurements is to use an external hardware setting that has a latency at a lower order. This complicates scalability investigation, as well as the use of testbeds installed in labs. Developing testbeds facilitating external hardware connection and allowing for low level and interrupt handling programming will be useful.

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 (Sec. I-B). 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 (Fig. 2), and MLE for offset/skew estimation in Gaussian model [1]. Remember that nodes exchange two-way messages to construct a quadruple of timestamps that forms a sample. The process is then repeated for a certain rounds to acquire a set of samples for estimation. The skew-offset model captures the clock drifting, such that once the synchronization parameters are estimated, they are used for a certain period before re-synchronizing the nodes.

We varied two important parameters of the implementation, the sample size, say k, i.e. the number of rounds before estimation, and the period separating two rounds, say T. Synchronization error has been measured for each variant of the implementation, where $t = 0$ is the time when the estimation is completed and the protocol execution is stopped for every variant (similarly to III-A). Three values for k have been used (10, 15, 20), and two for T ($50msec$, $100msec$). The results presented in Fig 6 confirm that the increase of k improvers the stability of the estimator over time as reported in all theoretical approaches. But more importantly, the results show that the parameter T has also a significant impact. For all values of k, the precision of every variant with $T = 50$ (dashed lines) has better performance than the respective one with $T = 100$ (solid lines), e.g., the variant $(20, 50)$ outperforms $(20, 100)$, etc. 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 σ^2 (the variance), 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 $T = 100$ in all scenarios, e.g. it was $1.30\mu sec^2$ for $(20, 50)$ vs. $4.14\mu sec^2$ for $(20, 100)$. 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 over longer periods. This 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 phases at the beginning of every cycle. The synchronization phase can be at the beginning of the cycle (which is the most common), or even split throughout the cycle. This is to avoid long latencies for data packets. The sample size should be fixed according to the required precision. For instance, if we consider the simple scenario used in the experiment reported in this section (just for illustration), a $1msec$ precision can be satisfied with $k = 10$, $T = 50msec$, and re-synchronization period of $1500sec$, or with $k = 20$, $T = 50msec$, and re-synchronization period of $3000sec$, etc.

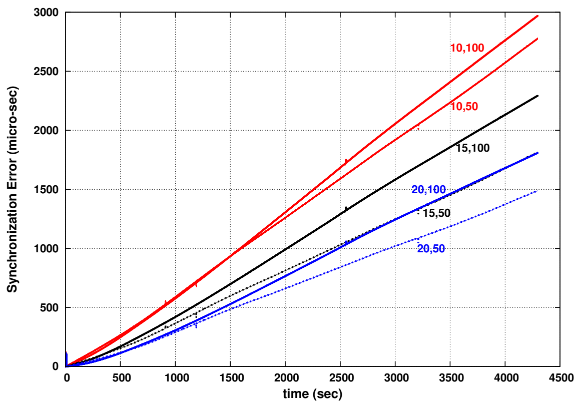

Fig. 6: Impact of Empirical Parameters

V. Protocol Implementations

A. Overview

The most relevant implementations of synchronization protocols from the literature are discussed in this section. A taxonomy of these protocols is presented in Fig. 7. Protocols framed in rectangles implements the joint skew/offset model, while those framed in diamonds are limited to the offset-only model. Normal lines for the frames represent implementation at the micro-second level, while those with dashed lines represent low-precision implementations that generally use low-granularity clocks. Some protocols use higher-layer timestamping and have the advantage of large portability, while those using low-layer timestamping reduce the impact of the delay variation at the cost of reducing the portability. All the protocols implement multi-hop synchronization, except RRTE and RATS. Multi-hop synchronization are divided into two categories: i) tree-based synchronization, where all the nodes are periodically synchronized to a single reference (usually the sink node), and ii) flat synchronization, where nodes are synchronized in peer-to-peer way. The second class is more suitable for distributed networks and event-based applications, where periodic maintenance of the tree maybe inappropriate. Protocols of the first class are centralized and use a single node as a reference that launches the synchronization protocol, and to which nodes synchronize. They can be appropriate for periodic applications, but maintaining such a tree and global synchronization is not justified in event-based applications. Many solutions build multi-hop synchronization as an extension to the single-hop synchronization. Some protocols have what is known as the gradient property, where high precision is targeted for local (single-hop) synchronization, and less importance is attached to the precision between remote nodes. TinyOS is the most popular software platform (operating system), followed by Contiki, where Mica motes family is the most popular hardware platform.

B. Offset-only Solutions

The offset-only model has the advantage of simplifying estimator calculation. However, frequent re-synchronization is required to keep high precision. Note that the precisions reported in what follows reflect measurements obtained immediately after synchronization before clocks start drifting, i.e., one shot synchronization. The offset-only solutions are not useful for long-term synchronization. They can be used in applications that need 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. Descriptions and discussions in this paper are focusing on implementation issues. For detailed presentation of each protocol, the reader can refer to other surveys, e.g. [4], or to the original paper introducing the protocol.

1) Timing-Sync Protocol for Sensor Networks (TPSN): For TPSN [12] implementation, MICA motes have been used, whose crystal has a maximum frequency of $4Mhz$. This permits to achieve a granularity of $0.25\mu sec$. RBS [2] has also been implemented by the authors for comparison. MAC-layer timestamping has been used to reduce the impact of the delay variation. 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 used MAC-layer timestamping for TPSN, and application layer timestamping for RBS. This leads to an unfair comparison and questionable conclusions. RBS concept is independent from

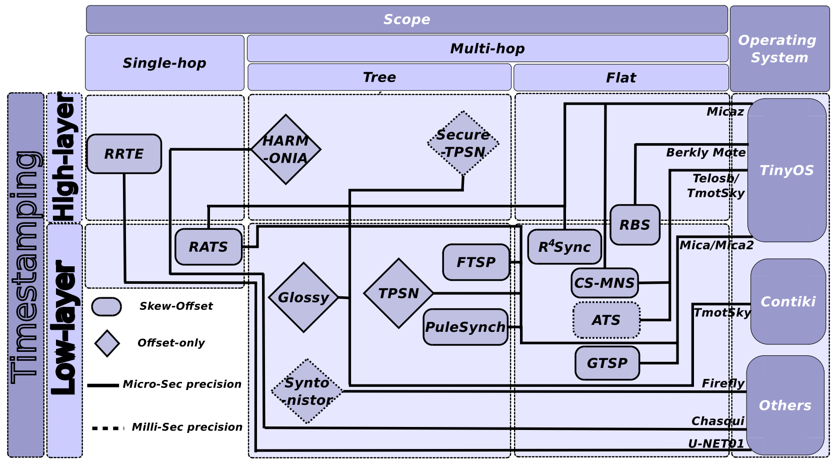

Fig. 7: Overview of the protocol implementations presented in the paper

the timestamping level, as it supports both high-layer and low-layer timestamping. Further, RBS low-layer timestamping has been implemented and tested, [2]. A fair comparison would use an implementation with the same timestamping layer, in which case RBS would be more accurate as it uses the receiver-to-receiver approach (Sec. II).

2) Secure-TPSN: Chen et al. [11] propose a secure TPSN-based synchronization protocol. The authors argument the use of $CC2420$ (TelosB's radio) by the supporting of hardware-based $128-bit$ AES encryption for their security operations, which aim at protecting synchronization signals from malicious attacks. The authors deployed time synchronization as a service at the application layer, but with timestamping at the MAC layer. The low-resolution external clock has been used, with a tick frequency set to $512Hz$ (instead of the maximum $32.768KHz$ enabled by the crystal), i.e. permitting a resolution of about $2msec$ per tick. Eleven nodes have been used in the experiment, arranged in single-hop, as well as multi-hop topologies. Results show that the synchronization error has been at the same order for single-hop and multi-hop scenarios, as well as for synchronization with and without security. It was between one and three ticks, with an average below $1.5tick$. This is basically due to the MAC-layer timestamping that completely eliminates the effect of the message variability in this experiment, given the very weak accuracy of the used clock.

3) HARMONIA: HARMONIA [13] is a simple synchronization protocol proposed for a wastewater monitoring and actuation application (CSOnet). In this application, nodes are supposed to wake up $6sec$ each $5min$ for collecting data and synchronizing to the sink (2% duty cycle). At every cycle, synchronization is used to coordinate the next cycle and assure nodes will wake up at the same time (to a certain degree of precision). Two different clocks are used; an external clock (RCC) that has low drift but also low resolution ($1sec$ granularity), and the microcontroller clock (MCC) that provides high resolution ($0.125\mu sec$) but suffers from high drifting and does not operate in the sleep mode. To tackle the high drifting of the MCC, the idea behind HARMONIA is to use the latter to accurately synchronize RCC, which will be used for duty-cycling the nodes. The MCC is synchronized within one active slot, and then it is used to set the RCC at all nodes along the tree to that of the sink. This use of a stable clock justifies the the offset-only model. A commercialized hardware (Chasqui node) is used. It is a modified version of MICA2 with a closed-source MAC protocol, which justifies the application layer timestamping.

In the experiment, five nodes plus a sink node have been used in different topologies including some multi-hop ones. In addition to the synchronization precision, the synchronization time has been measured, which is defined as the time to synchronizing all nodes to the sink. This represents the most important metric for the proposed protocol given the targeted objective of fast synchronization. HARMONIA has been compared with FTSP, and the results show that the former demonstrates superiority in terms of synchronization time, where the latter provides higher precision.

4) Glossy: Glossy [14] combines network message flooding and synchronization in a single protocol. It exploits constructive interference of IEEE $802.15.4$ packets and targets network wide synchronization. Similarly to HARMONIA, two clocks have been used; the high-frequency DCO and the low-frequency external crystal. 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. This is to implement duty-cycle and coordinate flooding. Influenced by FTSP (that will be presented later), Glossy compensates software jitter by measuring the gap between interrupt reception and interrupt service, then accordingly inserting a certain number of non-operations (NOP) instructions 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 show an average error below $1\mu sec$, even for multi-hop synchronization. The higher precision of Glossy compared to HARMONIA is basically due to the jitter compensation at the interrupt level.

5) Syntonistor: Rowe et al. [15] present a hardware module for global clock synchronization called Syntonistor. It operates at $60Hz$ and allows nodes to be tuned to the magnetic field radiating from existing AC power lines. This hardware has been attached to *Firefly* motes in the experiment. The AC power forms a signal that can be used as a global clock source for battery operated sensor nodes. It permits reducing drift between nodes, compared to the use of internal clocks. Still, there is typically a phase-offset between nodes. A protocol is used to compensate for this phase-offset, where a master node (sink) broadcasts a message at its rising pulse. It timestamps the message at low-layer immediately before transmission. The message is flooded across the network using the $CC2420$ radio. Every sensor node maintains a timer to count for time since its last rising edge pulse. The evaluation of the solution demonstrated only millisecond level precision. The major advantage of this hardware is its power-efficiency. The authors claim that it consumes less than $58\mu W$, which is more than twenty times lower than the energy consumed by most MAC protocols when in idle mode. Taking advantage of high stability of the magnetic field phase resulting from the AC power line eliminates the need of frequent re-synchronization. However, this hardware solution has some drawbacks. In addition to low resolution, the authors reported none operation with mobile networks. This is due to the abundance of the magnetic field sources in all directions that prevents the hardware receiver from working properly. Another major limitation is the need of active power line near the device, which is unsuitable for some applications in remote and hostile locations, or during a power outage. Table I summarizes the protocols presented in this subsection with respect to the influencing features presented in Sec III. For abbreviation, MTS stands for MAC-layer timestamping.

TABLE I

Protocol	Fast drifting	Delay variation	Fault-tolerance	Security
TPSN	No	MTS	No	No
Sec-TPSN	No	MTS	No	Yes
Harmonia	2 clocks	No	No	No
Glossy	2 clock	NOP insertion	No	No
Syntonistor	AC power	MTS	No	No

C. Joint Skew/Offset Solutions

Implementations that involve skew estimation are presented in this section. The skew estimation allows for long-term synchronization and reduces the impact of the clock drifting, but it has higher memory footprint and computation cost compared to the offset-only solutions.

1) Reference Broadcast Synchronization (RBS): RBS [2] is the first protocol that introduced 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 protocols proposed ahead of RBS. 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 offset (motivated by the confirmed zero mean Gaussian distribution of the delay differences). Further, the protocol supports both high layer timestamping and low-layer timestamping. RBS has also been implemented and tested with Berkeley mote, as well as commodity hardware platform running UNIX daemon with UDP datagrams. This implementation has been carried out for testing low-layer timestamping that was not possible with TinyOS and the Berkely mote of that period [2]. Results for high-layer timestamping show error at the order of few micro seconds. Those with low-layer timestamping 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 five 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. This confirms that the high level of abstraction in the conception with regard to timestamping does not eliminate the possibility of low-layer timestamping. This aspect has been ignored by many protocols proposed ahead of RBS, which wrongly assume RBS only supports high-layer timestamping, e.g. TPSN [12].

2) Flooding Time Synchronization Protocol (FTSP): Maroti et al. [6] propose FTSP, the first protocol that considers interrupt jitter compensation. This is by recording several timestamps per packet at the sender and receiver sides, then averaging and normalizing the obtained timestamps to come out with a final timestamp of the outgoing message. 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 errors 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 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 $CC2420$, 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. It also deviates from the central concept of FTSP that consists in using multiple-stamping. For comparison at the microsecond level with FTSP, a careful modification of FTSP implementation is then required.

3) Gradient Time Synchronization Protocol (GTSP): Sommer et al. [16] focus on the local synchronization (between direct neighboring nodes) 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, then logical skew and offset are calculated at each node using simple averaging. A formal proof of convergence is provided. $Timer3$ of ATmega128L has been used in the implementation. It operates at $1/8$ of the external oscillator frequency, i.e., at $921kHz$. The authors argue that packet-oriented radio chips, e.g., $CC2420$ (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, e.g. MAC scheduling.

4) Average Time Synchronization (ATS): Similarly to GTSP, Schenato and Fiorenti [7] propose a referenceless distributed solution that uses the logical (virtual) clock principle, where all nodes exchange their clock values and the averaged values are considered as consensus. Formal convergence has been proved by assuming the absence of communication delay, which is unrealistic in WSN. In the implementation, the low-resolution external clock of TmoteSky has been used at $32.768KHz$, i.e., with a resolution of $30.5\mu sec$ per tick. The implementation took advantage of the low-layer timestamping enabled by the $CC2420$ radio. 35 motes ranged in a grid topology have been used for the tests and comparison with the official distribution of FTSP. Results show synchronization error below 3 ticks for ATS, vs. errors up to 10 ticks for FTSP. The results can be argued by the fact that the TinyOS distribution of FTSP does not implement software jitter compensation technique (Sec. V-C2).

5) Rate Adaptive Time Synchronization (RATS): Ganeriwal et al. [17] propose a single-hop synchronization scheme to build a MAC protocol (UBMAC) that aims at minimizing uncertainty for preamble management. Ordinary least square regression has been used in the estimation to derive relative parameters. For prediction of error estimation, an interesting combination of analytical and empirical techniques has been used. The 'optimum' window size for estimation has been fixed empirically using two MICA2 motes in different scenarios, then historical error prediction has been analytically derived. An interesting feature of RATS is that it adapts the sampling period to the user specification on the synchronization bound. Experimental results demonstrate that the predictive duty-cycling provided by RATS gives important energy saving compared to the long-preamble of B-MAC (when fixing the error's higher-bound to the millisecond level). Further, preliminary results on RATS demonstrated errors below $3\mu sec$, but this level of granularity has not been used in UBMAC.

6) PulseSynch: Lenzen et al. [18] probabilistically analyze GTSP and FTSP, and the lower-bound of their respective skews. This analysis motivated their contribution on proposing a root square order bound algorithm. PulseSynch aims at distributing information on clock values as fast as possible, and focusing on multi-hop synchronization. Similarly to FTSP, a root node periodically floods its clock value through the network, but the delay jitter is added using the forwarder's estimate of the root advance instead of using local clock. Linear regression is used for skew estimation to mitigate the fast clock drifting. PulseSynch has been evaluated in a 20-node testbed, and compared to FTSP.

Acknowledgment of synchronization packets has been used at the application layer to eliminate their loss that is not tolerable by the protocol. Despite its cost in terms of communication overhead, this ACK mechanism is more realistic compared to the no-loss assumption. Timestamping of the first six bytes has been used to overcome the jitter. Results show fast convergence of pulseSynch and low errors at the order of few micro-seconds, a bit lower than FTSP in the tested scenarios.

7) Round-Robin Timing Exchange (RRTE): Huang and Wu [8] propose RRTE as a hybrid solution between TPSN and PBS [1]. In each cycle, one node is selected in a round-robin way to act as a second reference and execute PBS, while the other nodes overhear transmissions. This is to reduce the power consumption and balance it among all the nodes, instead of using a fixed second reference as in PBS. Recursive second order regression is used for clock adjustment and skew estimation, to compensate for the fast clock drifting. RRTE has been implemented and tested in U-NET01 platform, which features a U-NET01 8051 micro-controller with an embedded processor, a UBEC's $UZ2400$ wireless transceiver module that is IEEE 802.15.4 compliant. Results show precision of the order of several tens to several hundreds of microseconds. Comparison with implementations of TPSN and PBS on the same platform shows that RRTE balances between them, where TPSN demonstrated the best precision.

8) CS-MNS: In [19], Kunz and MacNeil describe an implementation of their previously proposed protocol called CS-MNS. It is a multi-hop mutual synchronization protocol, where nodes align their clock without turning to a reference. Motivated by the halt of the micro-controller internal clock in sleep mode, the external $32.768KHz$ clock has been used. The hardware radio packet timestamping capabilities of the underlying platform has been used. Clock adjustments were implemented using fixed-point arithmetic on 64 bits, with eight bits for the whole part and 56 bits to the right of the radix point. This is to avoid the use of software floating-point libraries that may yield a significant memory footprint. Scenarios of up to 14 nodes have been investigated, and CS-MNS has been compared with FTSP. Results show superiority of CS-MNS, but with synchronization error of several tens of microseconds, and up to few milliseconds. This looks completely contradictory compared to the results reported by FTSP. Still, this can be justified by the use of the low-resolution clock in the TinyOS official distribution of FTSP (Sec. V-C4).

9) R^4Syn: R^4Syn [9] is a distributed receiver-to-receiver protocol, where all nodes cooperatively assure the traditional role of the reference in a round-robin way. Beacons and timestamp exchanges are integrated in the same steps to accelerate sample acquisition and estimator calculation. Djenouri et al. [10] provide a fault-tolerant implementation and a testbed experimentation of R^{4}Syn. Seven MICAz motes have been used and a software topology for multi-hop scenarios. Time has been implemented using the internal high-resolution clock. Synchronizing motes have been connected via available *pins* to an Arduino that periodically triggers measurement. R^{4}Syn allows both low-layer and high-layer timestamping, but only low-layer timestamping has been evaluated. Both skew-offset and offset-only models have been implemented. MLE has been used for skew estimation along with MAC-layer timestamping. Extensive tests show errors at the order of few microseconds; below $3\mu sec$ on average for single hop, and $20\mu sec$ for 6-hop scenarios. 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.

An interesting result illustrated in the experiment is the comparison between the offset-only and skew-offset models, similarly to the experiment presented in Sec. III-A. This gives a clear picture on the skew estimation impact. A major lack in the experiment is with regard to high-layer timestamping, which is an important feature of R^{4}Syn. Table II summarizes the protocols presented in this subsection. For abbreviation, MTS stands for MAC-layer timestamping, ST for the several timestamping per packet technique (Sec V-C2), ES for the use of estimation methods, and R2R for the receiver-to-receiver approach.

TABLE II

Protocol	Fast drifting	Delay variation	Fault-tolerance	Security
RBS	ES	MTS+R2R	No	No
FTSP	ES	MTS+ST	No	No
GTSP	ES	MTS+ST	No	No
ATS	ES	NO	No	No
RATS	ES	MTS	No	No
PulseSynch	ES	ES+ST	No	No
RRTE	ES	No	No	No
CS-MNS	ES	MTS	No	No
R4Syn	ES	MTS+R2R	Yes	No

VI. Discussions

One of the most influencing features that impact a synchronization protocol is the clock drifting. Skew estimation models represent the vital solution to this problem, which have been used in the solutions presented in Sec. V-C. Nonetheless, their implementation comes at a higher complexity compared to the offset-only model. Our experiments with MICAz motes confirm suitability of linear models for skew estimation, both for internal and external clocks as reported in the paper (Sec. III). The internal clock does not run when in sleep mode, and thus it cannot be used in applications such as coordinated duty-cycling. The VHT technique used by HARMOIA and Glossy enables the use of the high-granularity internal clocks to accurately synchronize the external clocks, then to make a precise rendezvous with the external clocks. This technique is useful in applications that need precise rendezvous or synchronized coordinated actions, such as coordinated duty-cycling (e.g. TDMA-like scheduling), but not those needing time measurement or conversion (timestamping), e.g., object tracking. Offset-only implementations do not permit long-term synchronization. They can only be used in applications requiring sporadic delay-tolerant synchronization, where the synchronization can be performed instantly before the timestamping of the reported event.

Another big challenge is the delay for message exchange and handling, which is prone to high variability. We showed that parameters of the synchronization protocol such as the delay between sample acquisition (beaconing) considerably affect the delay variance, and thus the estimators' precision (Sec. IV). MAC-layer timestamping is a practical solution, but comes at a reduced portability. One of the practical approaches in real implementation is to trading-off the synchronization precision for a reduced cost, as in RATS. The precision should be bounded according to the application needs. Another practical technique 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. the one of MICA2, and not the common packet oriented radios, e.g. $CC2420$. Finally, we realized that many protocols have been compared with the TinyOS distribution of FTSP that uses millisecond interfaces. This does not implement the multiple timestamping per packet, which is FTSP's main contribution. For acurate investigation, a more fair comparison would use byte-level interrupt triggering platforms and a revisited implementation.

VII. Conclusion

Implementation of synchronization protocols in wireless sensor networks (WSN) has been considered in this paper. Practical aspects that have been neglected or ideally abstracted in the theoretical solutions have been investigated in this paper. After introducing some general concepts and backgrounds, the challenges that any implementation of a synchronization protocol in WSN faces have been listed. Clock drifting and delay variation are the most influencing features. A Taxonomy of the state-of-the-art implemented protocols have been given before describing the protocols. Descriptions in this paper have been implementation-centric, where the focus was on practical aspects related to the implementation and experimentation. Algorithmic description has been largely covered in the literature and is out of the scope of this paper. Solutions that are limited to the offset-only model have been first presented, followed by those using the joint skew-offset model for drift compensation. While the former class has the advantage of simplicity, the latter provides long-term synchronization. Techniques to compensate for the jitter (delay variation) such as low-layer timestamping, the use of normalized several timestamps per packet, and the virtual-high resolution time have been discussed. Some unfair comparisons with FTSP and RBS protocols have reported. We hope this moderate work will be useful for researchers interested into any issue relate to implementation of synchronization protocols in WSN.

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] 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.

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

[4] 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.

[5] J. Zheng and Y.-C. Wu, "Joint time synchronization and localization of an unknown node in wireless sensor networks," *Trans. Sig. Proc.*, vol. 58, no. 3, pp. 1309–1320, 2010.

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

[7] L. Schenato and F. Fiorentin, "Average timesynch: A consensus-based protocol for clock synchronization in wireless sensor networks," *Automatica*, vol. 47, no. 9, pp. 1878–1886, 2011.

[8] Y.-H. Huang and S.-H. Wu, "Time synchronization protocol for small-scale wireless sensor networks," in *IEEE Wireless Communications and Networking Conference (WCNC'10)*, 2010, pp. 1–5.

[9] 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.

[10] 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.

[11] S. Chen, A. Dunkels, F. sterlind, T. Voigt, and M. Johansson, "Time synchronization for predictable and secure data collection in wireless sensor networks," in *The Sixth Annual Mediterranean Ad Hoc Networking WorkShop*, 2007, pp. 165–172.

[12] 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.

[13] 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.

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

[15] A. Rowe, V. Gupta, and R. Rajkumar, "Low-power clock synchronization using electromagnetic energy radiating from AC power lines," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys'09)*, 2009, pp. 211–224.

[16] 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.

[17] S. Ganeriwal, I. Tsigkogiannis, H. Shim, V. Tsiatsis, M. B. Srivastava, and D. Ganesan, "Estimating clock uncertainty for efficient duty-cycling in sensor networks," *IEEE/ACM Transations on Networking.*, vol. 17, pp. 843–856, June 2009.

[18] C. Lenzen, P. Sommer, and R. Wattenhofer, "Optimal clock synchronization in networks," in *Proceedings of the 7th ACM International Conference on Embedded Networked Sensor Systems (SenSys'09)*, November 2009, pp. 225–238.

[19] T. Kunz and E. McKnight-MacNeil, "Implementing clock synchronization in wsn: Cs-mns vs. ftsp," in *WiMob*, 2011, pp. 157–164.