Fault-Tolerant Implementation of a Distributed MLE-based Time Synchronization Protocol for Wireless Sensor Networks

Djamel Djenouri[1], Nassima Merabtine[2], Fatma Zohra Mekahlia[2], and Messaoud Doudou[1]
1: CERIST Research Center, Algiers, Algeria
2: Saad Dahlab University, Blida, Algeria
Email: ddjenouri@acm.org, nassimane@gmail.com, mekahlia.fzohra@yahoo.fr, doudou@mail.cerist.dz

***Abstract*—This paper describes the implementation and evaluation of R^4Syn protocol on MICAz platform and TinyOS operating system. The contribution is two folds. First, the implementation uses thorough maximum-likelihood estimators (MLE) in the joint offset/skew model, while all similar MLE-based estimators are merely evaluated with theoretical and numerical analysis thus far, and empirical solutions use simple computation estimators, such as offset-only models, or linear regression for skew estimation. Difficulties that has been encountered and overcome are reported in this paper. The second contribution is to consider fault-tolerance, an aspect that has been completely abstracted in previous works. The implementation assures correct behavior despite nodes failure or packet loss, as demonstrated by the experiments. Experimental results also demonstrate microsecond-level precision and long-term validity of the estimators in the joint skew/offset model.**

I. Introduction

Wireless sensor networks (WSN) are generally designed for physical world monitoring, where sensors nodes perform coordinated tasks with possible actuation on the physical world. Most of the tasks require nodes to be synchronized to assure thorough progress. For instance, sensor readings from multiple nodes need to be combined (aggregation, fusion, averaging, etc.) either to synthesize a snapshot of the conditions in time, or to track the progress of a phenomenon through the monitored field. Medium access scheduling is another example where synchronization is vital for correct functioning. Applications involving realtime actuation are also examples where fine grained synchronization is of high importance. Generally speaking, time synchronization can be defined as the service for generating a mutually consistent timestamping service across the nodes of a WSN. This may be achieved either through the use of common reference(s), or the use of relative synchronization mechanisms enabling to convert local clock readings from a node to another.

The lack of a common clock and shared memory make message-exchange-based algorithms the only way to ensure synchronization in WSN, and generally speaking, in distributed systems. The variability of the exchanged messages latency raises the need of appropriate and accurate estimators. The problem is more complex in wireless networks, where communication may be prone to high delay variability. Further, WSN node limitations (computation, memory, energy) add more constraints to the problem. Several time synchronization protocols have been proposed thus far [1]. Some are designed theoretically and evaluated either by simulation or mathematical analysis, while others are empirical and focus on implementation on sensor motes. Most of the empirical studies focus on practical problems, and use simplified estimators to facilitate the implementation. Theoretical studies– including our previously proposed protocol, R4syn– use more elaborated estimators, but completely neglect practical aspects one may face for real implementation. The work presented in this paper can be viewed as empirical one, but with the focus on implementing thorough maximum likelihood estimators (MLE) on tiny motes, to provide long-life high precision synchronization despite node limitations. It consists in implementing and evaluating R^{4syn} protocol [2] in real sensor motes, while considering all the practical issues related to node limitations that has been abstracted in the previous paper [2].

The rest of the paper is organized as follows, the next section summarizes the related work and provides the background, Section III describes the implementation, followed by the experimental results in Section IV. Finally, Section V concludes the paper and sketches the perspectives.

II. Related Work and Background

Time synchronization in wireless sensor networks is a topic that attracted the focus of researchers in the last few years. [3] and [4] are good introductive surveys on this topic, where [1] and [5] provide up-to-date reviews. Many solutions focus on practical aspects and evaluate the proposed protocols in testbeds of real sensor motes. Examples of such empirical work include [6], [7], [8], [9], [6], [10], [11], [12]. Several other solutions are more theoretical, such as [13], [14], [15], [16], which define thorough mathematical estimators/models to provide fine-grained long-life synchronization. Long life synchronization aims at ensuring synchronization parameters (estimators) to be valid with a high degree of precision long a long time before the synchronization protocol is re-executed. However, these works are evaluated either by simulation or through theoretical mathematical models without any real implementation, where they make implicit abstraction of practical

aspects, such as limitation of sensor motes. The aim of this work is to tackle this aspect, and implement our previously proposed protocol, R^4syn, with its ML estimators, in sensor motes. The implementation considers further practical challenges, such as fault-tolerance.

R^4syn [2] is based upon RBS [12], which introduces the receiver-to-receiver approach and exploits the broadcast property of the wireless communication medium. That is, receivers located within listening distance of the same sender receive a broadcast message at approximately the same time, with little variability due to the reception timestamping latency at the receivers. This property reduces the time-critical path, which is defined as the path of a message that contributes to non-deterministic errors in a protocol [3]. In other words, the sender's delay variability is eliminated with the receiver-to-receiver concept (compared to traditional sender-to-receiver solutions). In RBS, a sequence of synchronization messages (beacons) are periodically transmitted from a given and fixed sender- termed reference- and intercepted by all synchronizing nodes. The nodes timestamp the arrival-time of each beacon using their local clocks, then every pair of nodes exchange the recorded timestamps to construct samples. If the i^{th} beacon is timestamped, u_i, by node, n_1, and v_i, by node, n_2, the pair (u_i, v_i) forms a sample. These samples are used to estimate the *relative* offset and skew between nodes, n_1, and n_2.

Similarly to RBS, R^4syn uses relative receiver-to-receiver synchronization, where every node has its own clock that runs independently from the others. The synchronization is ensured by estimating parameters reflecting relative deviation with respect to every other node, where no local-clock update is needed. But contrary to RBS, no reference or super node is needed; all nodes are sensor motes that cooperatively get synchronized. Nodes are assumed to be neighborhood-aware, i.e., each node knows the ID of every neighboring node. The RBS reference role is equally divided amongst all participating nodes, while beacons and timestamp exchange are merged in the same packets/steps. The proposed protocol runs in cycles, where nodes sequentially broadcast enhanced beacons, i.e. beacons carrying timestamps. IDs can be used to determine the order of the sequence (logical ring). A beacon carries timestamps that report local reception times of previous beacons. For a neighborhood of K nodes, every beacon would carry $K-1$ timestamps, which precipitates sample acquisition. Maximum likelihood estimators (MLE) have been proposed for both the offset-only and joint skew/offset models, and the synchronization parameters between any couple of nodes are estimated in the two models by Eq. 1, Eq. 2 and 3, respectively. Where $\widehat{\theta}_{mle}$ is the offset estimator in the offset-only model, and $\widehat{\alpha}_{mle}$, $\widehat{\beta}_{mle}$ are respectively the skew and offset estimators in the joint skew/offset model. (u_i, v_i) are the samples acquired through the protocol execution. Detailed description of R^4syn is available in [2].

$$\widehat{\theta}_{mle} = \frac{\sum_{i=1}^{K}(u_i - v_i)}{K} \tag{1}$$

$$\widehat{\alpha}_{mle} = \frac{\sum_{i=1}^{K} u_i \sum_{i=1}^{K} v_i - K \sum_{i=1}^{K} v_i u_i}{(\sum_{i=1}^{K} v_i)^2 - K \sum_{i=1}^{K} v_i^2}, \tag{2}$$

$$\widehat{\beta}_{mle} = \frac{1}{K}(\sum_{i=1}^{K} u_i - \widehat{\alpha}_{mle} \sum_{i=1}^{K} v_i) =$$

$$\frac{1}{K}(\sum_{i=1}^{K} u_i - \frac{\sum_{i=1}^{K} u_i \sum_{i=1}^{K} v_i - K \sum_{i=1}^{K} v_i u_i}{(\sum_{i=1}^{K} v_i)^2 - K \sum_{i=1}^{K} v_i^2} \sum_{i=1}^{K} v_i). \tag{3}$$

III. Implementation

MICAz motes have been used, along with TinyOS as the underlying operating system [17]. MICAz has an IEEE 802.15.4 radio (CC2420), which is a ZigBee compliant RF transceiver that operates in the Industrial scientific and medical (ISM) band, at 2.4 to 2.4834 GHz. MICAz has low power consumption and uses two AA batteries. Its micro-controller is an $8-bit$, 8 MHz ATmega128L, with a 4 KB RAM and 128 KB external flash memory. The mote includes two clocks; i) an internal clock with a high granularity, but also with a drift at the order of few tens PPM (up to 100PPM) [18], and ii) a more stable external crystal clock but with $32khz$ frequency, i.e. enabling a tick (granularity) of about 32 μsec. Although being more stable, the later does not allow for a microsecond level precision. The former clock has been used in the experiment to investigate high precision feasibility. If only a lower precision satisfies the application requirements (e.g. a precision at the order of milliseconds or tens of microseconds), it is straight forward to use the current implementation with the other clock. An Arduino has been used for measuring the synchronization error. This microcontroller is an $8-bit$, $16Mhz$ ATmega328P, with 2 KB RAM, and 32 KB flash memory.

Implementation of the estimators proposed in Eq.1 and Eq. 2 has been the major challenge we faced. Theses formulas include multiplications of very large numbers (clock values), which cause variable overflow after few seconds due to memory, TinyOS, and the mote logical and arithmetic unit limitations that does not permit the manipulation of large numbers. This prevents getting large samples, and thus achieving microsecond level precision would not be possible. To tackle this, the formulas have been rewritten. It can be noticed that both the numerator and denominator of Eq. 2 are the difference of two large terms, but the final differences are small numbers that do not cause any computation problem for division. The problem is caused by the calculation of the

large intermediate terms. For the numerator, the first term can be seen as the sum of products ($u_i v_j$), consisting of k^2 elementary terms, and the second is the sum of products $u_j v_j$ that repeats K times, resulting in k^2 elementary terms as well. Therefore, instead of calculating the two large terms of the nominator separately then subtracting them, it is more practical to subtracting from each elementary term of the first term, $u_i v_j$, an elementary term of the second one, $u_j v_j$. The same principle is applied to the denominator. Formally speaking, Eq. 2 is rewritten as follows,

$$\widehat{\alpha}_{mle} = \frac{\sum_{j=1}^{K}\sum_{i=1}^{K} u_j v_i - \sum_{j=1}^{K}\sum_{i=1}^{K} u_i v_i}{\sum_{j=1}^{K}\sum_{i=1}^{K} v_j v_i - \sum_{j=1}^{K}\sum_{i=1}^{K} v_i^2} = \frac{\sum_{j=1}^{K}\sum_{i=1}^{K}(u_j v_i - u_i v_i)}{\sum_{j=1}^{K}\sum_{i=1}^{K}(v_j v_i - v_i^2)}, \quad (4)$$

The difference of large products are then transformed into the sum of differences of small products, which does not cause any problem for the implementation. The resulted estimator of α is replaced in Eq. 3 to obtain the new expression of β estimator.

$$\widehat{\beta}_{mle} = \frac{1}{K}(\sum_{i=1}^{K} u_i - \widehat{\alpha}_{mle} \sum_{i=1}^{K} v_i). \quad (5)$$

After this reformulation, it was possible to implement the estimators. In addition to adapting the estimators, the implementation takes into account possible node failure. Timeout mechanisms have been added, such that when a node is perceived not reporting its beacon, then the next one can broadcast its beacon without waiting for the predecessor in the logical ring. Further, if the node is perceived not to reporting a beacon up to a certain number of cycles, the schedule will be updated accordingly, and the faulty node is removed. Also note that the float computation is not supported by MICAz hardware; it is implemented as a library at the kernel in TinyOS 2. Several problems have been faced with this library when handling large numbers (clock values). We then re-implemented float division calculation. It was possible to achieve high degree of synchronization with just 5 digits for the fraction part.

The size of the binary code has been 17600 bytes in ROM, 984 bytes in RAM for the offset-only model, and 28470 bytes in ROM, 1434 bytes in RAM for the skew/offset model. This includes code for protocol signaling, estimator calculations, the operating system kernel and libraries, and control operations for measurement, all uploaded as a single file. In the experiments, three nodes running the protocol have been used, two of them have been plugged to an Arduino through available *pins* for precision measurement (Fig. 1). At regular intervals after few seconds of warm-up phase allowing estimators to converge, the Arduino transmits a signal to both motes. At interrupt handler for the appropriate *pin*, one of the mote picks up its clock, while the other estimates it using the estimators resulted from the execution of the protocol. Both are connected to a laptop through a serial port for constructing log files. The experiment has been conducted using both offset-only and joint skew/offset models.

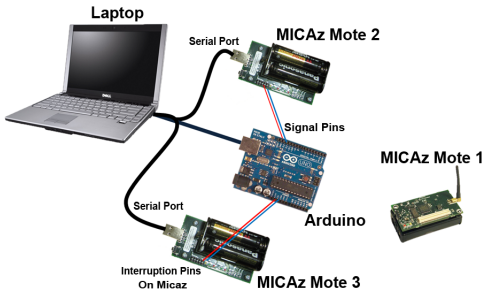

Fig. 1. Experiment Setup

IV. Experimental Results

A. Precision Evaluation

Fig. 2 and Fig. 3 plot the synchronization error for the offset-only model, skew/offset model respectively. They show very similar results, where error ranges from $1\mu sec$ to $7\mu sec$, with most values between $3\mu sec$ and $5\mu sec$. Each plot has been obtained after one continuous execution. Some variability has been noted when repeating the experiment, which is typical in WSN due to instability of wireless channels, environment, sensor motes' contexts, etc. All these parameters affect delay variability, and clocks drifting. We performed 14 times the same test at different environmental conditions. Fig. 4 depict the average values, where the error bars are plotted with 95% of confidence interval. Results show average values below $10\mu sec$ most of the time, with a few microseconds variability .

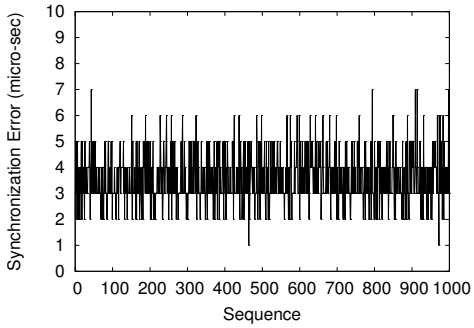

Fig. 2. Synchronization error, offset-only Model

B. Stability Evaluation

The results reported thus far are recorded when measurements are launched right after estimators are calculated, with

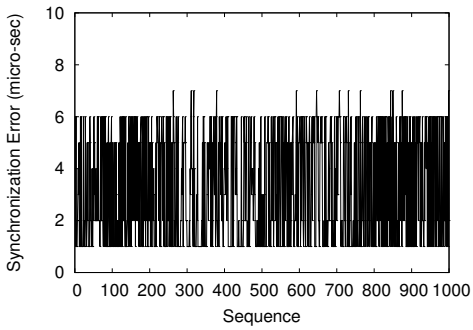

Fig. 3. Synchronization error, offset/skew Model

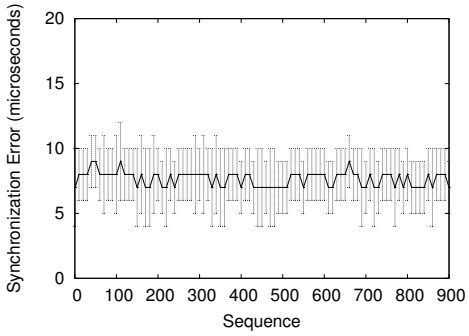

Fig. 4. Averaged Synchronization error, Offset/skew Model

a possible latency of few microseconds. In real deployment, however, estimators should generally be used for some period before updating them. The longest the period is, the more stable and efficient the protocol is considered. This enables reducing the frequency of signaling and thus the overhead, while keeping the high precision. To investigate this issue, the protocol execution has been stopped once estimators convergence after the warm up period. Measures are then taken from the moment the protocol run is halted. Fig. 5 draws the synchronization error vs. time for both models. The offset-only model that does not consider clock drifts causes fast degradation of the precision. For instance, the error exceeds $500\mu sec$ after $1min$, $1msec$ after $2min$, and $6msec$ after $10min$. On the other hand, the skew/offset model assures more stability through predicting the drift. Its precision remains at the order of few microseconds (less than $10\mu sec$) during more than $200sec$, and it does not exceed $330\mu sec$ during the whole $15min$ experiment duration. These results confirm the joint model to be more stable and effective for general WSN applications. The offset-only model has the simplicity advantage, and it may be useful when implemented with more stable clocks (like the external crystal clock), provided that only low precision is required (e.g. at the millisecond order).

C. Fault-tolerance Evaluation

In the following, the implemented protocol behavior in presence of failed nodes is investigated. Six motes have been used (node 0 through node 5), and failures of some nodes

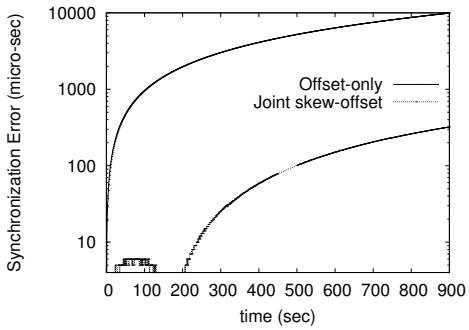

Fig. 5. Estimators accuracy over time

have been simulated by turning them off at scheduled intervals. Protocol functioning has been monitored by analyzing control packets at a base-station. Node 1 is switched off at time $10sec$, then node 2 at time $20sec$, and node 3 at time $30sec$. All the nodes are tuned-on later at time $40sec$. Fig. 6 shows that every correct node continuous broadcasting beacons despite the failure of its predecessor. It also shows that the nodes are reintegrated and resume broadcasting beacons after they are turned on. The subsequent synchronization error is plotted in Fig. 7. Node failure causes the reduction of the number of samples, which inevitable affects the precision. Nevertheless, the increase in error is smooth, and it does not exceed $20\mu sec$ when three nodes are down. More importantly, the precision returns to its lowest values when the nodes are turned-on.

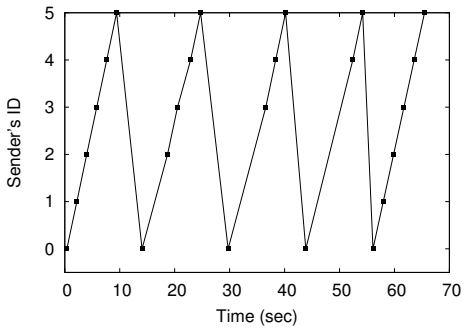

Fig. 6. Protocol execution with faulty nodes

V. Conclusion and Perspectives

This paper dealt with implementation of a synchronization protocol on real sensor motes. This is a challenging problem given the limitation of sensor motes, and the resource consuming operations of time synchronization procedures, notably estimator calculation. Our previously proposed protocol, $R4syn$, has been implemented on micaZ motes and TinyOS operating system. The implementation adds a fault-tolerance module to support node failures; an aspect neglected by the first version of the protocol. Formulas of estimators have been rewritten to prevent variable overflow problems. The

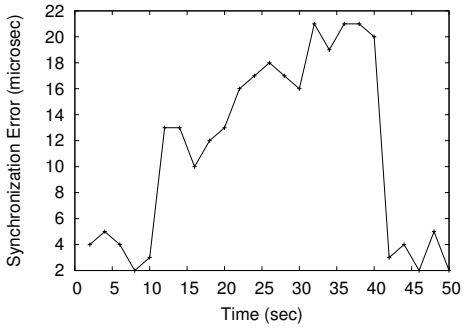

Fig. 7. Synchronization error with faulty nodes

state-of-the-art empirical studies implementing synchronization protocols use simplified estimators, such as offset-only model, or linear regression for skew estimation. On contrary, maximum likelihood estimators for the joint skew/offset model are implemented in this work. To our knowledge, this work is the first that implements such thorough estimators on real sensor motes. $R4syn$ implementation has been tested, and experimental results demonstrate microsecond precision, where the synchronization error has not exceeded few microseconds in both offset-only and skew/offset models. The skew/offset model– which captures clock drifts– demonstrated more stability, where the estimators have much longer lifetime and remain effective in keeping the microsecond-level precision for several minutes. The offset-only model has the simplicity advantage, and it may be useful when implemented with more stable clocks (like the external crystal clock), provided that only low precision is required (e.g. at the order of milliseconds). Fault-tolerance has also been investigated and confirmed.

Several perspective yields from this work. The clock used in the implementation is turned-off when in sleep mode, which would affect the protocol progress. Application of Virtual High-resolution Time technique (VHT) [7] to this implementation is interesting. The VHT technique consists in using external clock as a reference to estimate sleep periods and switching to the high resolution clock for time conversion when in active mode. Multi-hop implementation is also among the perspectives.

REFERENCES

[1] O. Simeone, U. Spagnolini, Y. Bar-Ness, and S. H. Strogatz, "Distributed synchronization in wireless networks," *IEEE Signal Processing Magazine*, vol. 25, no. 5, pp. 81–97, 2008.

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

[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] C. Lenzen, T. Locher, P. Sommer, and R. Wattenhofer, "Clock synchronization: Open problems in theory and practice," in *Proceedings of the 36th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '10)*, ser. LNCS 5901. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 61–70.

[6] 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.

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

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

[9] 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.

[10] S. Yoon, C. Veerarittiphan, and M. L. Sichitiu, "Tiny-sync: Tight time synchronization for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 3, no. 2, Jun. 2007.

[11] 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.

[12] 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.

[13] I. Sari, E. Serpedin, K.-L. Noh, Q. M. Chaudhari, and B. W. Suter, "On the joint synchronization of clock offset and skew in rbs-protocol," *IEEE Transactions on Communications*, vol. 56, no. 5, pp. 700–703, 2008.

[14] M. Leng and Y.-C. Wu, "On clock synchronization algorithms for wireless sensor networks under unknown delay," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 1, pp. 182–190, 2010.

[15] 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, March 2010.

[16] K.-L. Noh, Q. M. Chaudhari, E. Serpedin, and B. W. Suter, "Novel clock phase offset and skew estimation using two-way timing message exchanges for wireless sensor networks," *IEEE Transactions on Communications*, vol. 55, no. 4, pp. 766–777, 2007.

[17] P. Levis and D. Gay, *TinyOs Programming*. Cambridge University Press, 2009.

[18] F. Ferrari, A. Meier, and L. Thiele, "Secondis: An adaptive dissemination protocol for synchronizing wireless sensor networks," in *Proceedings of the 7th IEEE Conference on Sensor Mesh and Ad Hoc Communications and Networks (SECON'10)*, June 2010, pp. 1–9.