

Fault-Tolerant WSN Time Synchronization*

Ung-Jin Jang, Sung-Gu Lee, Jun-Young Park, Sung-Joo Yoo

Department of Electronic and Electrical Engineering, Pohang University of Science and Technology, Pohang, Korea

E-mail: {itmind, slee, reinhard, sungjoo.yoo}@postech.ac.kr

Received June 13, 2010; revised July 16, 2010; accepted August 21, 2010

Abstract

This paper proposes a new fault-tolerant time synchronization algorithm for wireless sensor networks that requires a short time for synchronization, achieves a guaranteed time synchronization level for all non-faulty nodes, accommodates nodes that enter suspended mode and then wake up, is computationally efficient, operates in a completely decentralized manner and tolerates up to f (out of $2f + 1$ total) faulty nodes. The performance of the proposed algorithm is analyzed, and an equation is derived for the resynchronization interval required for a specific level of synchronization precision. Results obtained from real runs on multi-hop networks are used to demonstrate the claimed features of the proposed algorithm.

Keywords: Fault Tolerance, Time Synchronization, Wireless Sensor Network

1. Introduction

A wireless sensor network (WSN) can be used to gather specific physical or chemical data in a target area. A sensor node typically uses a microprocessor with low speed and low power usage and a radio module with low data rates because most sensor nodes used in WSNs are low-cost devices powered by batteries. In spite of the limited capabilities of the sensor node, the WSN is still a very useful system for data gathering because it “autonomously” gathers data and maintains network connections until its nodes’ batteries are entirely consumed. For example, a WSN can be constructed to monitor the ecological environment for a long period without additional human intervention. A WSN is also useful for monitoring harsh environments such as fire areas, toxic-spill areas, and mine fields.

In some WSN applications, the sensor nodes require identical time values, *i.e.* synchronized clocks, with each other. For example, let us consider an application to detect and track a target using triangulation. If we use the time difference of arrival of sonic waves for target detection, time differences can be measured exactly when three nodes that hear the sound have synchronized clocks. Another useful application is time-division multiple access (TDMA) for wireless communication. By using

the TDMA method, each node of a WSN can communicate without radio channel conflicts, thereby reducing packet loss and energy consumption due to rebroadcasting. If the nodes don’t have synchronized clocks, TDMA communication on a fine scale cannot be used. Synchronized clocks are also required for periodic sleep-and-wake scheduling of sensor nodes. The sensor nodes go to sleep to reduce energy consumption when they have nothing to do; then they wake up when there is something to do such as sensing, transmitting, or receiving. To make sure that sufficient numbers of nodes remain awake in a given area at any one time, such sleep-wake cycles must be coordinated using synchronized clocks.

Time synchronization, or clock synchronization, refers to the means by which nodes in a distributed network are able to maintain clock values that are identical with each other within a specific error bound. Time synchronization has been studied by many researchers over the past 30+ years. Most of the previously studied works considered general computing system and wired communication environments. However, the computational capabilities of a node in a WSN are limited, and the wireless communication environment is more unpredictable and unreliable than a wired communication environment. So, such works can’t be directly adopted for use in WSNs. In recent years, a number of time synchronization methods have been specifically proposed for use with WSNs. Reference Broadcast Synchronization (RBS) [1], Time-sync Protocol for Sensor Networks (TPSN) [2] and Flooding Time Synchronization Protocol (FTSP) [3] are three of

*A very preliminary version of this work was presented at the 2008 International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC). This paper is a major rewrite of that version that includes basic changes to the proposed algorithm, more detailed analysis and additional experimental results.

the most well-known and practical methods that have been proposed. RBS uses receiver-receiver synchronization, in which a reference node broadcasts an event packet to let receivers compare time differences with each other. By having receivers compare the times when they received the same broadcast event packet, it is possible to mitigate the effect of widely variable communication delays in wireless networks. TPSN uses a tree structure and adopts MAC-layer timestamps for accurate synchronization, whereas FTSP uses flooding and more precise MAC-layer timestamps to improve performance. These methods are all highly dependent on the reference node selected, and there are no mechanisms for dealing with timer faults or otherwise faulty nodes.

Since WSNs typically use large numbers of low-cost devices, fault-tolerance should be of utmost concern with any WSN application. Recently, a few works have addressed the fault-tolerance problem with regards to WSN time synchronization. However, methods such as [4] and [5] only work in single-hop networks, while other methods such as [6-8] considered multi-hop networks with severe restrictions on node behavior and/or limited topologies.

In this paper, fault-tolerance is addressed by proposing a practical and generally usable fault model (based on experiments with actual WSN devices), and a time synchronization method is proposed that can achieve a guaranteed level of fault-tolerance in arbitrary multi-hop networks. The proposed method, Fast-Fault Tolerant Synchronization (FFTS), tolerates up to f faulty nodes while guaranteeing a specific level of maximum synchronization error. This paper is organized as follows. Section 2 gives the system model and Section 3 describes the proposed time synchronization method, FFTS. Section 4 provides an analysis of the FFTS algorithm and Section 5 gives experimental results. Finally, Section 6 provides concluding remarks.

2. System Model

The system model used is based on our experience with practical WSN systems built using current commercial devices, such as MICAz motes. Let us consider a network N with n sensor nodes. Each node s_i ($i = 1, 2, \dots, n$) has a hardware counter (also referred to timer), which is increased at a rate based on a hardware oscillator. The value of counter $c_i(t)$ at real time t is used to represent a local time value $tv_i(t)$,

$$tv_i(t) = \frac{c_i(t)}{FO_i} \quad (1)$$

where FO_i is the frequency of the local timer. If the exact value of FO_i is known and this value remains constant, then time synchronization between two nodes can be achieved simply by a one-time adjustment of the effec-

tive timer frequency and initial offset of one node to follow that of the other node. However, in real hardware devices, the value of FO_i varies over time due to environmental factors and intrinsic hardware variations. To model this type of behavior, let the drift rate ρ_i of s_i denote the difference between the current local time value and the real time value. Then, the local time value can be expressed as (2). The local time value increases faster than real time when ρ_i is positive and increases slower than real time when ρ_i is negative.

$$tv_i(t) = (1 + \rho_i) \cdot t \quad (2)$$

In a general WSN N , the local time values of the sensor nodes will tend to drift away from other. Thus, in order to maintain a global time base, a time synchronization procedure needs to be executed at each node. Let us consider the use of a procedure in which each node periodically adjusts the frequency and offset of its timer based on the local time value of a unique reference node, as communicated to it through a multi-hop wireless network.

“Real time” is extremely difficult to maintain as real time is an “amorphous concept” whose exact value depends on the movement of celestial objects in space. Thus, real time is normally approximated by an internationally agreed-upon standard time referred to as Coordinated Universal Time (UTC) [9]. However, even accurate UTC time is difficult to maintain as it requires a radio receiver that receives periodic radio broadcasts of UTC time. With the low-cost sensor nodes typically used in WSNs, it is assumed that such radio receivers are not used. Given this limitation, time synchronization in the WSN depends heavily on the unique reference node used. This reference node should be a “nonfaulty” node with small variability in the drift rate of its timer. If we regard t to be the reference time value (the local time of the reference node), the synchronized time value $tv_sync_i(t)$ can be obtained by adjusting the drift rate and initial offset of the local time of node s_i relative to the reference node. If the node s_i has been resynchronized at times t_1 and t_2 (in that sequence), then the synchronized time at time t after t_2 can be approximated using the following formula:

$$tv_sync_i(t) = f_{adj} \cdot t_{elapsed} + t_{offset} \quad (3)$$

where

$$f_{adj} = \frac{t_2 - t_1}{tv_i(t_2) - tv_i(t_1)},$$

$$t_{elapsed} = tv_i(t) - tv_i(t_1),$$

$$t_{offset} = tv_i(t_1)$$

Due to cost limitations, each sensor node of a WSN is assumed to have low-cost timer, processor, and wireless communication modules. This will result in high node failure rates and wide variations in the quality of timer modules. The use of fault tolerance techniques can sig-

nificantly increase the reliability and accuracy of applications executed on such WSNs. Thus, in this paper, we propose a fault-tolerant time synchronization method based on message-passing. To analyze the expected behavior of our method, the set of faults to be tolerated will be restricted to timer faults as defined below. If nodes can fail in any manner, then time synchronization cannot be achieved for nonfaulty nodes that can communicate with the reference node only through such faulty nodes. However, it is expected that most faults in WSNs will involve nodes that either stop working altogether (due to depletion of batteries) or produce highly inaccurate timer values. “Fail-stop” nodes can be tolerated by commonly-used fault-tolerance techniques that involve routing around such nodes. Thus, the fault-tolerance techniques presented in this paper will focus on timer faults.

Definition 1: Given a permitted drift rate ρ , a node s_i is nonfaulty during the real time interval $[t_1, t_2]$ if $(1 - \rho)(t_2 - t_1) < tv_i(t_2) - tv_i(t_1) < (1 + \rho)(t_2 - t_1)$. Otherwise, s_i has a timer fault.

A node with a timer fault will be considered to be a *faulty* node. Since the local time maintained by each node will be adjusted periodically according to Equation (3), a faulty node will be one in which the drift rate has changed drastically from the last time it was measured. A node reporting a local time value drastically different from neighboring nodes can also be diagnosed as faulty by those neighboring nodes based on Definition 1. A node with a timer fault should definitely *not* be selected as the reference node. The level of synchronization to be achieved and the problem to be solved in this paper can be written as follows.

Definition 2: Given a synchronization bound δ , two nodes s_i, s_j are δ -synchronized at real time t if and only if $|tv_i(t) - tv_j(t)| \leq \delta$ [10].

Fault-Tolerant Time Synchronization Problem: For a given WSN with $2f + 1$ nodes, of which up to f nodes may be faulty, devise a software procedure that can be executed at each nonfaulty node such that all nonfaulty nodes remain δ -synchronized during the lifetime of the WSN.

3. Fast Fault-Tolerant Time Synchronization

The proposed *Fast Fault-tolerant Time Synchronization (FFTS)* algorithm, executes iteratively in two phases. In Phase 1, executed by a node when it first starts time synchronization and when it wakes up after being in sleep mode (in order to conserve its energy) for a long time, a short resynchronization interval P1 is used. After Phase 1 has been executed for a fixed number of iterations (after K SYNC packets have been received), the node will enter Phase 2 and use a much longer resyn-

chronization interval P2. The use of a short resynchronization interval during Phase 1 permits a node to quickly become synchronized to a reference node. The use of a longer resynchronization interval during Phase 2 permits a node to maintain time synchronization with low messaging and computing overhead. Except for the resynchronization interval used, the operation of a node during Phase 1 and Phase 2 are identical.

Within each phase of the FFTS algorithm, the nodes in the WSN first cooperate with each other to decide on a reference node. After the reference node has been determined, each node adjusts its local time iteratively (with period P1 or P2) based on Equation (3).

The main “intelligence” of the FFTS algorithm involves the selection of the reference node. Given the possibility of up to f faulty nodes, a node should collect at least $f + 1$ timestamp values since up to f of these values may be faulty. However, for $f + 1$ timestamps to be sufficient, there must be an indicator of which of these $f + 1$ timestamps are nonfaulty (such as a nonforgeable signature). Since such a nonfaulty indicator is not assumed, $2f + 1$ timestamps are used. The nonfaulty timestamp can then simply be selected as the median value, which *must* be nonfaulty by Definition 1.

The possibility of multi-hop message transmissions in WSNs presents a challenge when attempting to select a unique nonfaulty reference node for time synchronization purposes. If timestamps are broadcast by all nodes in a periodic manner and each node waits until $2f + 1$ timestamps are collected before selecting a median value, several such median values will exist in a multi-hop network since areas of the network that are widely separated will tend to form separate cliques. All such median values must be reconciled into a unique reference timestamp in order to realize a time-synchronized network. The approach adopted in the FFTS algorithm is to combine two median values into the maximum of the two values whenever two such values are observed (a “max-of-median” approach). After a few iterations (with the number of iterations dependent on the network diameter, the maximum distance in hops between any two nodes of the network), this will result in a globally unique reference timestamp (the timestamp generated by the reference node) value.

Two types of packets are used to implement the above procedure. INITSYNC packets contain up to $2f + 1$ time values (ID and timestamp pairs). If a node receives an INITSYNC packet that contains less than $2f + 1$ time value, the node appends its time value to an empty time value field of the INITSYNC packet and rebroadcasts the INITSYNC packet. However, since time elapses during this action, the node must also add the elapsed time to the previously recorded time values before rebroadcast-

ing the INITSYNC packet. Once an INITSYNC with $2f + 1$ values is received, a SYNC packet, which only requires only time value field, is sent out with the median value. If a node receives two SYNC packets with different timestamps, the larger value is chosen as the unique reference and used to form subsequent SYNC packets. Eventually, all nodes will receive a SYNC packet having the unique reference time value. Then the local time of each node can be adjusted using Equation (3) and the timestamp of the reference node.

Besides the two main techniques discussed above (the use of two different synchronization intervals and max-of-median reference time selection), several other methods are used to enhance the performance of the FFTS algorithm. When creating the timestamps to be included in INITSYNC and SYNC packets, the timestamps are created *immediately before* the packets are transmitted at the Medium Access Control (MAC) layer of the software. Such MAC-level timestamps [2,3] result in highly accurate communication delay estimates since the highly variable channel access delays are not included in the communication delay estimates. Also, before each INITSYNC or SYNC packet is transmitted, a random backoff delay is used. If an INITSYNC or SYNC packet is received before the random backoff delay expires, the random backoff is canceled and the received packet is processed first. This is a method that is commonly used in wireless networks with independently operating nodes since this reduces packet collisions and permits coordinated behavior using independently operating nodes. For details on the implementation of the FFTS algorithm as described above, the reader is referred to the pseudocode description given in **Figure 1**. This pseudocode shows the procedure to be executed by each node in the WSN N .

4. Analysis

Let us first form an analysis of the FFTS algorithm for one-hop time synchronization and then extend that analysis to the general multi-hop case. In FFTS, time synchronization is performed by adjusting the local timer frequency (f_{adj}) and the local timer offset (t_{offset}) (refer to Section 2) based on the timestamp in a SYNC packet. This is done periodically with period T ($T = P1$ or $P2$). Thus, differences in the global time maintained by two nodes result from changes in the drift rates of the two nodes over the time period T . In [1], Elson *et al.* showed, based on analysis and experiments, that the differences in the times when two nodes receive a packet broadcast over a wireless medium follows a Gaussian distribution with zero mean and σ standard deviation. Thus, for a

Algorithm: FFTS

```

Repeat for each node  $s_i$  in  $N$ :
1: If  $iter < K$ 
2:   Wait for P1;
3: else
4:   Wait for P2;
5:    $iter \leftarrow iter + 1$ ;
6:   Wait for a random backoff delay;
7:   If a INITSYNC is received during back-off
8:     If timestamps in the INITSYNC  $< 2f + 1$ 
9:       Append  $tv_i(t)$  to the INITSYNC;
10:    Adjust all timestamps in INITSYNC to
        account for time elapsed thus far in node  $s_i$ ;
11:    Broadcast updated INITSYNC;
12:   Else /*  $2f + 1$  timestamp values present */
13:     Select median timestamp value;
14:     Broadcast SYNC packet with median value;
15:   Else if a SYNC (with reference  $r$ ) is received at time  $tv_i(t)$ 
16:     If the SYNC is the first SYNC in this period or  $tv_i(t) > tv_{ref}(t)$ 
        /* max-of-median */
17:        $ref \leftarrow r$ ;
18:        $f_{adj} = Drift\_Adjust(tv_i(t), tv_{ref}(t))$ ;
19:       Broadcast updated SYNC;
20:     Else
21:       Drop the SYNC;
22:   Else /* no packet is received during backoff */
23:     Send INITSYNC with  $tv_i(t)$ ;

```

Figure 1. FFTS algorithm.

Procedure: *Drift_Adjust*

Input: receive time $tv_r(t)$, send time $tv_s(t)$

Output: adjusted drift-rate

```

1: If  $tv_r(t)$  is the first received time value for  $r$ 
2:    $t_{1-r} \leftarrow tv_r(t)$ ;
3:    $t_{offset} \leftarrow tv_s(t)$ ;
4:   return 0;
5: Else
6:   return  $\frac{tv_r(t) - t_{1-r}}{tv_s(t) - t_{offset}}$ ;

```

Figure 2. *Drift_Adjust* procedure.

one-hop network, we have the following useful theorem.

Theorem 1: Given a maximum change in timer drift-rate of ρ' , the maximum synchronization error δ among non-faulty nodes in a one-hop network is given by the following Gaussian distribution.

$$\delta \sim N(2\rho'T, 2(1 + \rho')\sigma^2)$$

Proof: The exact resynchronization (drift-rate adjustment) periods observed by different non-faulty nodes in a one-hop network are T plus or minus t_i , where t_i is a term determined by INITSYNC packet exchange times, random back-off times, CPU processing times and variations in SYNC packet reception times. Of these components in t_i , FFTS compensates for all measurable delays incurred (and appends the corresponding drift-rate-adjusted timestamp to the SYNC packet) before a SYNC packet is sent. Thus, since variations in packet reception times are not compensated in FFTS, the resynchronization periods observed by non-faulty nodes follow a Gaussian distribution with mean T and variance σ^2 . Let s_1

be the node that experiences the largest positive change in timer drift during one resynchronization period. Then, after one period, s_1 will have a synchronization error described by $X_1 \sim N(\rho'_1 T, (1 + \rho'_1) \sigma^2)$ [11]. Likewise, if s_2 is the node that experiences the largest negative change in timer drift during one period, s_2 's synchronization error will be $X_2 \sim N(-\rho'_2 T, (1 + \rho'_2) \sigma^2)$. Thus, since s_1 and s_2 are independent, $X_1 - X_2$ will also be a Gaussian distribution $N((\rho'_1 + \rho'_2) T, (2 + \rho'_1 + \rho'_2) \sigma^2)$ [11]. Substituting the maximum change in timer drift ρ' for both ρ'_1 and ρ'_2 , the theorem follows.

Let us now extend the above theorem to a multi-hop network. In [1], Elson *et al.* showed that the standard deviation of inter-receiver phase offsets increases by a factor of \sqrt{m} over the one-hop standard deviation, given a hop-distance of m . Thus, the following corollary is also true.

Corollary 1: Given an m -hop network, the maximum synchronization error δ is given by the following Gaussian distribution.

$$\delta \sim N(2\rho'T, 2m(1 + \rho')\sigma^2)$$

Corollary 1 can be used to derive the resynchronization interval $T = P2$ to be used to achieve a specific level of global time precision. A typical large value for ρ that we observed in experiments with MICAz WSN devices was around $100 \mu\text{s/s}$. However, the maximum *change* in the drift rate (ρ') that we observed in our experiments (refer to **Figure 3**) was around $0.16 \mu\text{s/s}$. The standard deviation σ of the differences in packet delivery times was observed to be around $4.65 \mu\text{s}$. Assuming a maximum permitted change in drift-rate, for non-faulty nodes, of $0.15 \mu\text{s/s}$ and $m = 5$, the resynchronization period required for $\delta = 50 \mu\text{s}$ (with 99% probability) was found to be $T = 52.5$ seconds. If $\delta = 55 \mu\text{s}$, $T = 69.2$ seconds, and if $\delta = 60 \mu\text{s}$, $T = 85.8$ seconds (both with 99% probability).

5. Experimental Results

In order to evaluate the proposed algorithm, experiments were conducted using 13 MICAz (Berkeley “mote”) WSN devices. MICAz uses an 8-bit ATmega128 microprocessor, a 7.3728 MHz clock and a CC2420 ZigBee radio. In the first set of experiments, shown in **Figure 3**, one MICAz device was used as a reference node, and the drift-rate changes of the timers used in the other 12 nodes were measured with respect to the reference node. The most severe drift rate change observed was approximately $0.16 \mu\text{s/s}$. The next most severe drift rate observed was about $0.12 \mu\text{s/s}$. Depending on the requirements of the application, the nonfaulty timer drift-rate change threshold could be set to a value such as $\rho' = 0.15 \mu\text{s/s}$ (to tolerate $0.16 \mu\text{s/s}$) or $\rho' = 0.1 \mu\text{s/s}$ (to tolerate $0.16 \mu\text{s/s}$ and $0.12 \mu\text{s/s}$).

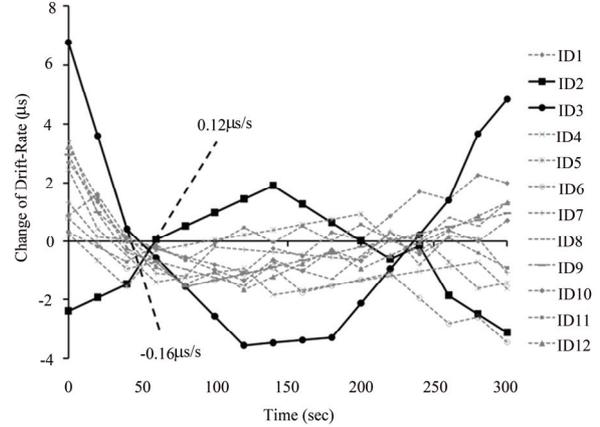


Figure 3. The drift-rate changes of 12 nodes. Black solid lines represent two nodes that have the most severe (ID2) and next most severe (ID3) drift-rate change and dotted gray lines represent the other nodes.

In the second set of experiments, the fault-tolerance characteristics of the FFTS algorithm were compared with the FTSP algorithm in a 12-node one-hop network. In this case, since the faults being tolerated are timer faults (timers with widely varying drift-rate changes), fault-tolerance implies a smaller drift away from the reference time value after a synchronization operation. FTSP was used for comparison purposes because it used a similar system model and had the best time synchronization performance of the previously proposed methods [3]. Since FTSP uses an arbitrary fixed node as its reference node, we experimented with several different fixed nodes as the reference node. FFTS was implemented with $f = 1$. The reference node used by FFTS changed several times during these experiments.

Experiments are conducted for FFTS and FTSP with different choices of reference nodes. Each experiment was repeated 100 times and average values recorded. **Figures 4** and **5** show the average and maximum, respectively, time synchronization errors (pair-wise time differences) observed in the five minutes after all nodes were synchronized. The results for the FTSP algorithm are summarized by showing the plots with the best, worst and an arbitrary intermediate choice for the reference node, determined in hindsight (labeled as FTSP_Best, FTSP_Worst, and FTSP_Arbitrary respectively). The experimental results show that the use of FFTS results in significantly lower synchronization error than FTSP. This is a byproduct of the fact that faulty nodes are defined to be those nodes with high drift-rate changes. As shown in **Figure 3**, drift rates of the timers for all nodes tend to change as time progresses. Thus, the FFTS policy of continually selecting the node with the median drift rate as the reference node tends to keep time values tightly clustered together. The lower synchronization error

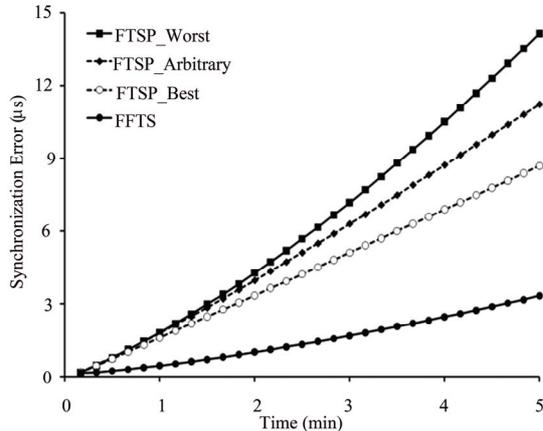


Figure 4. Average pair-wise time offsets of 12-node during 5-minute period with various reference node selection methods. Black solid line with black circles represents synchronization error of FFTS. Solid line with squares, dotted line with diamonds and dotted line with white circles represent FTSP_Worst, FTSP_Arbitrary and FTSP_Best, respectively.

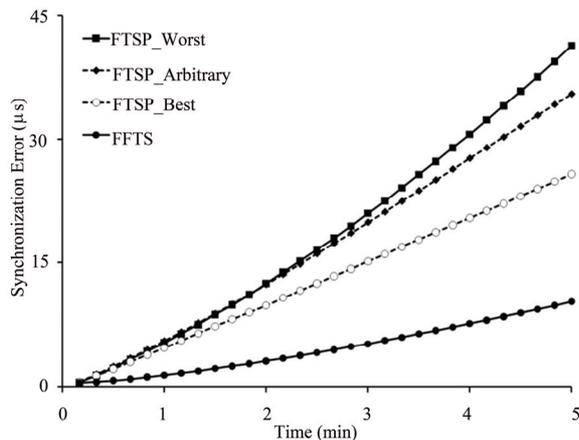


Figure 5. Maximum pair-wise time offsets of 12-node during 5-minute period with various reference node selection methods. Black solid line with black circles represents synchronization error of FFTS. Solid line with squares, dotted line with diamonds and dotted line with white circles represent FTSP_Worst, FTSP_Arbitrary and FTSP_Best, respectively.

of FFTS implies that FFTS can achieve the same time synchronization precision as FTSP while using a longer resynchronization interval, thereby also reducing energy usage.

In the third set of experiments, time synchronization was performed using the FFTS algorithm in multi-hop networks. The time synchronization precision achieved was quantified by measuring the average and maximum pair-wise offsets between every pair of nodes. The results of the multi-hop experiments are shown in **Figure 6**. A 3 x 4 2-dimensional mesh network topology was used

and P1 and P2 were set to 2 seconds and 30 seconds, respectively. K, the number of times when the short period P1 was used, was set to 6 (one more than the network diameter of a 3 x 4 mesh) and f was again set to 1. The drift-rate change threshold was set to $\rho' = 0.15 \mu\text{s/s}$. Using a target maximum time synchronization error of $45 \mu\text{s}$, the resynchronization interval required was 35.8 seconds. However, during the course of our experiments, it was discovered that operations within MICAz devices could not be timed exactly due to interrupts and slow CPU operation. Thus, a P2 value of 30 seconds was used.

There are several notable features about the results shown in **Figure 6**. First, the precision of the time synchronization achieved was about $20 \mu\text{s}$ throughout the 20-minute experiment. This was well within the δ value predicted by Corollary 1. Second, initial synchronization was completed in about $K \cdot P1 = 12$ seconds. Third, resynchronization of newly woken-up nodes was also very fast. At time A (about 8.6 min), we turned off 4 nodes in the third row of the 3 x 4 mesh. Then, when we turned on these nodes again at time B (about 14 min), these nodes were resynchronized within about 10 seconds. This fast initial synchronization and resynchronization are the result of the short time period P1 used.

These results can be compared with the simulation results shown for FTSP in [3]. The degree of precision achieved was about the same as the results obtained by FTSP [3], which used a similar experimental platform and had the best time synchronization results among the previously proposed methods. However, FFTS has the additional benefit of a quantified level of fault-tolerance. In addition, resynchronization (after being turned off and on) was very slow in FTSP (around 6 minutes in one set of experiments [3]) whereas it is much faster (on the order

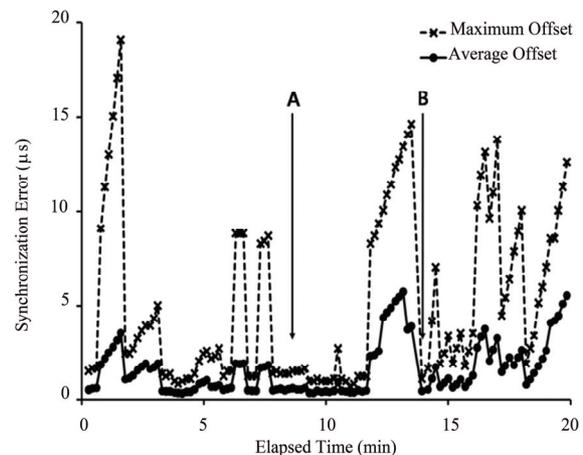


Figure 6. Average and maximum pair-wise time offsets in a 3 x 4 2D mesh-type network. Solid line represents average offset and dotted line represents maximum offset during 20 minutes.

of a few seconds) with FFTS.

6. Conclusions

This paper presents a system model and a new time synchronization algorithm, termed Fast Fault-tolerant Time Synchronization (FFTS), based on this model. Analysis is used to quantify the resynchronization interval required for a desired level of time synchronization precision and the protocol to be used to achieve a desired level of fault-tolerance. Experimental results are used to show that, in terms of the level of time precision achieved, the proposed algorithm works as well as the best previously proposed algorithms. The FFTS algorithm, however, works in a completely decentralized manner, thereby increasing its fault-tolerance and adaptability to changing network conditions, is able to tolerate faulty nodes (nodes with faulty timers) and achieves extremely fast resynchronization of nodes that wake up from suspended mode.

7. References

- [1] J. E. Elson, L. Girod and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts," *The 5th Symposium on Operating Systems Design and Implementation*, Vol. 36, No. Winter 2002, 2002, pp. 147-163.
- [2] S. Ganeriwal, R. Kumar and M. B. Srivastava, "Timing-Sync Protocol for Sensor Networks," *Proceeding of the 1st International Conference on Embedded Networked Sensor System*, Los Angeles, 2003, pp. 138-149.
- [3] M. Maroti, B. Kusy, G. Simon and A. Ledeczi, "The Flooding Time Synchronization Protocol," *Proceeding of the 2nd International Conference on Embedded Networked Sensor Systems*, Baltimore, 2004, pp. 39-49.
- [4] P. N. K. Sun and C. Wang, "Fault-Tolerant Cluster-Wise Clock Synchronization for Wireless Sensor Networks," *IEEE Transactions on Dependable and Secure Computing*, Vol. 2, No. 3, 2005, pp. 177-189.
- [5] S. Z. H. Song and G. Cao, "Attack-Resilient Time Synchronization for Wireless Sensor Networks," *Proceeding of the International Conference on Mobile Adhoc and Sensor Systems*, Washington D.C., 2005, pp. 765-772.
- [6] S. Hwang and Y. Baek, "Fault Tolerant Time Synchronization for Wireless Sensor Networks," *Lecture Note in Computer Science*, Vol. 3894, 2006, pp. 480-493.
- [7] P. N. K. Sun and C. Wang, "Secure and Resilient Clock Synchronization in Wireless Sensor Networks," *IEEE Transactions on Selected Areas in Communications*, Vol. 24, No. 2, 2006, pp. 395-408.
- [8] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks," *IEEE Transactions on Computers*, Vol. 55, No. 2, 2006, pp. 214-226.
- [9] D. Mills, "Internet Time Synchronization: The Network Time Protocol," *IEEE Transactions on Communications*, Vol. 39, No. 10, 1991, pp. 1482-1493.
- [10] P. Ramathan, K. G. Shin and R. W. Butler, "Fault-Tolerant Clock Synchronization in Distributed Systems," *Computer*, Vol. 23, No. 10, 1990, pp. 33-42.
- [11] A. Papoulis and S. U. Pillai, "Probability, Random Variables and Stochastic Processes," McGraw-Hill, New York, 2002.