Time Synchronization Based On Slow Flooding in Wireless Sensor Networks

Kasım Sinan Yıldırım and Aylin Kantarcı

Abstract—The accurate and efficient operation of many applications and protocols in wireless sensor networks require synchronized notion of time. In order to achieve network-wide time synchronization, a common strategy is to flood current time information of a reference node into the network, which is utilized by the de facto time synchronization protocol Flooding Time Synchronization Protocol (FTSP). In FTSP, the propagation speed of the flood is slow since each node waits for a given period of time in order to propagate its time information about the reference node. It has been shown that slow-flooding decreases the synchronization accuracy and scalability of FTSP drastically. Alternatively, rapid-flooding approach is proposed in the literature, which allows nodes to propagate time information as quickly as possible. However, rapid flooding is difficult and has several drawbacks in wireless sensor networks. In this paper, our aim is to reduce the undesired effect of slow-flooding on the synchronization accuracy without changing the propagation speed of the flood. Within this context, we realize that the smaller the difference between the speeds of the clocks, the smaller the undesired effect of waiting times on the synchronization accuracy. In the light of this realization, our main contribution is to show that the synchronization accuracy and scalability of slow-flooding can drastically be improved by employing a clock speed agreement algorithm among the sensor nodes. We present an evaluation of this strategy on a testbed setup including 20 MICAz sensor nodes. Our theoretical findings and experimental results show that employing a clock speed agreement algorithm among the sensor nodes drastically improves the synchronization accuracy and scalability of slow-flooding.

Index Terms—Distributed Algorithms, Time Synchronization, Slow Flooding, Least-Squares, Clock Speed Agreemeent.

1 Introduction

Sensor nodes in wireless sensor networks (WSNs) are equipped with cheap hardware clocks which frequently drift apart due to their low-end quartz crystals. Since the drift can be different for each sensor node, the hardware clocks of the nodes may not remain always synchronized although they might have been synchronized when they are started up. Lack of synchronized time leads to inaccurate and inefficient operation of many applications and protocols in WSNs [1], [2]. Hence, a time synchronization protocol is required so that all nodes exchange their time information to synchronize their clocks for minimizing their synchronization error, i.e. *clock skew*.

A common method in order to achieve network-wide time synchronization in WSNs is to flood current time information of a reference node into the network. Flooding Time Synchronization Protocol (FTSP) [3], the de facto time synchronization protocol in WSN world, utilizes this method by allowing nodes to propagate their time information about the reference node after waiting for a given period of time. It has been shown that *slow-flooding* decreases the synchronization accuracy and scalability drastically due to the waiting times at each node [4]. In order to prevent these side effects, PulseSync [4] offers *rapid-flooding* by allowing nodes to propagate their time information quickly and reliably, and hence increasing the propagation speed of the flood.

However, although rapid flooding strategy improves the synchronization accuracy and scalability of flooding based time synchronization dramatically, it has several drawbacks. First, rapid-flooding in WSNs can also be slow due to neighborhood contention since the nodes cannot propagate the flood until their neighbors have finished their transmissions [5], [6]. Secondly, the transmissions of the sensor nodes need to be scheduled to avoid neighbor contention [4], [5], [7]. While this scheduling is straightforward for simple line topologies as used in the experiments in [4], it can be quite sophisticated in dense networks with other topologies. Finally, reliable rapid flooding in sensor networks is difficult due to packet losses. Retransmission, as a simple solution to recover lost packets, may lead to broadcast storm problem. More sophisticated solutions for loss recovery may decrease the speed of the flood and increase energy consumption of the nodes [8].

Consequently, the severe drawbacks of rapid flooding mentioned above bring us to the following question: Is it also possible to achieve scalable and tight synchronization in WSNs with slow flooding? In this paper, we answer this question positively by showing that the undesired effect of slow-flooding on the synchronization accuracy can be eliminated by preserving the propagation speed of the flood. For this purpose, we first consider FTSP and show that its drift estimation mechanism which estimates the progress speed of the reference node's clock with respect to the hardware clock speed of the other nodes exhibits an

• *The authors are with the Department of Computer Engineering, Ege University, İzmir, TURKEY, 35100.*
E-mail: {sinan.yildirim,aylin.kantarci}@ege.edu.tr

exponentially growing error with the network diameter. The waiting times at each hop due to slow-flooding amplify this error and degrades the accuracy of synchronization, leading to a serious scalability problem. Thus, we realize that the smaller the error of the drift estimation and hence the difference between the speeds of the clocks, the smaller the undesired effect of waiting times on the synchronization accuracy. In the light of this realization, our main contribution is to show that the synchronization accuracy and scalability of slow-flooding can drastically be improved by employing a clock speed agreement algorithm among sensor nodes. We introduce Flooding With Clock Speed Agreement (FCSA) protocol which forces all nodes to run at the same speed by employing an agreement algorithm and synchronizes them to a reference node which floods stable time for the whole network. We theoretically show that the synchronization error of FCSA grows with the square root of the network diameter, not exponentially as in FTSP.

There are synchronization protocols in the literature which also force all nodes to agree on a common clock speed [9], [10]. However, FCSA is different from these protocols in the following aspect. FCSA utilizes flooding together with clock speed agreement in order to optimize the synchronization error between the reference node and the remaining nodes in the network. This approach leads to a significant advantage: FCSA can be used for synchronizing sensor network to a stable time source such as such as Coordinated Universal Time (UTC), i.e. for *external synchronization*. On the other hand, Average TimeSynch (ATS) [10] and Gradient Time Synchronization Protocol (GTSP) [9] are designed to optimize the synchronization error between the neighboring nodes. This optimization is achieved in a completely decentralized way without employing flooding: Nodes in the network synchronize to their neighbors and there is not any special node which acts as a time reference. Unfortunately, this approach lacks the ability to provide synchronization to a stable time source [11], [6]. In order to achieve external synchronization in WSNs, flooding becomes essential and a reference node which is synchronized to an external time source is required. By flooding time information of the reference node and internally synchronizing the remaining sensor nodes to this node, network-wide synchronization can be established.

We implemented FCSA on a real WSN platform and compared its performance to FTSP, PulseSync and GTSP. Experimental results collected from a line topology of 20 sensor nodes show that FCSA improves the performance of GTSP remarkably and outperforms FTSP by a factor of 26 in terms of synchronization error. Moreover, we observed that FCSA exhibits a similar performance to PulseSync on the line topology, but it is superior on the grid topology where contention and congestion are not negligible. Our simulations showed that these results also hold on larger and denser networks.

The remainder of this paper is organized as follows. We provide related work and our system model in Section 2 and 3, respectively. We analyze the effect of slow-flooding in FTSP in Section 4. We present FCSA protocol in Section 5. Implementation details and experimental results are given in Section 6. Simulation results are presented in Section 7. Finally, Section 8 is the conclusion.

2 Related Work

There are considerable amount of protocol based studies in the literature which focus on minimizing synchronization errors in WSNs [12], [13], [14], [15], [3], [16], [17], [4], [11], [6]. Among these protocols, Flooding Time-Synchronization Protocol (FTSP) [3], the de facto standard time synchronization protocol in sensor networks, is designed to minimize the skew between any two nodes in the network, i.e. *global skew*. In FTSP, a dynamically elected reference node periodically floods the value of its clock (current time information) into the network. Using the flooded information, each slave node uses least-squares regression in order to establish a linear relationship between its hardware clock and the clock of the reference node. Using this relationship, each sensor node can predict future clock values of the reference node without communicating frequently. The nodes broadcast their predicted clock values of the reference node to their neighbors for the synchronization of the whole network. However, the predicted time information is not broadcasted quickly upon receiving a new synchronization message. Instead, each node waits until the expiration of its broadcast period. Due to the slow propagation speed of the flood arising from the waiting times, the estimation error of least-squares is amplified at each hop [4].

Lenzen et al. [4] proposed PulseSync protocol which employs rapid-flooding in order to reduce the effect of waiting times on the synchronization accuracy. PulseSync propagates time information from a reference node as fast as possible. Apart from the *rapid flooding* approach, each node uses a linear regression table to estimate the clock of the reference node, as in FTSP. Although this protocol improves the performance of FTSP drastically, rapid-flooding has several drawbacks [4], [5], [6], [7], as we mentioned previously. Hence, optimizing the performance of time synchronization protocols which use slow-flooding without changing the propagation speed of the flood and its message complexity appears as an important open problem.

The method of forcing all nodes to agree on a common clock speed, which we also employ in this paper, has been introduced with Average TimeSynch (ATS) [10] protocol. In addition to ATS, Gradient Time Synchronization Protocol (GTSP) [9] whose main objective is to minimize the clock skew between neighboring nodes, i.e. *local skew*, also employs this mechanism. Both ATS and GTSP do not require a

reference node: All nodes in the network synchronize to their neighbors. The clock values received from neighboring nodes are used to agree on a common clock value and clock speed. However, these studies do not focus on the synchronization to a reference node, which may be crucial if synchronization to a stable time source (or a real-time through a GPS receiver) is required by the sensor nodes [6].

3 System Model

We model a WSN as a graph $G = (V, E)$ where $V = \{1, ..., n\}$ and $E \subseteq V \times V$ represent the set of n sensor nodes with unique identifiers and the set of bidirectional communication links between the nodes, respectively. Any node $u \in V$ can communicate with any other node $v \in V$ to which it is directly connected and these nodes are referred as the neighbors of node u. $\mathcal{N}_u = \{v \in V | \{u, v\} \in E\}$ and $|\mathcal{N}_u|$ represent the set of neighbors and the number of neighbors of node u, respectively. The *distance* between any nodes u and v, denoted by $d(u, v)$, is defined as the number of edges on the shortest path between those two nodes in the graph G. The *diameter* of the graph G is the maximum distance between any two nodes, which is denoted by $\mathcal{D}$.

Each sensor node u is assumed to be equipped with an unmodifiable *hardware clock* $H_u()$. The reading of the hardware clock of the node u at real time t is defined as

$$H_u(t) = \int_0^t h_u(\tau) d\tau \quad (1)$$

where $h_u(\tau)$ represents the *rate* (speed) of the hardware clock at time τ. External crystal oscillators in sensor nodes are used as the clock source for their hardware clocks and the frequencies of these oscillators exhibit a drift between 30 and 100 ppm[1]. Therefore, it is assumed that hardware clocks have *bounded drifts* such that for all times t it holds that $1 - \varepsilon \leq h_u(t) \leq 1 + \varepsilon$ where $0 < \varepsilon \ll 1$. Since the hardware clocks of the nodes drift apart and the nodes cannot modify their hardware clocks, alternatively each node u maintains a *logical clock* $L_u()$ in order to acquire synchronized notion of time. The value of the logical clock is calculated by considering the hardware clock of that node and the information carried by the synchronization messages received. The *rate* of the logical clock L_u at time t is defined as

$$\frac{dL_u(t)}{dt} = l_u(t) \frac{dH_u(t)}{dt} = l_u(t) h_u(t) \quad (2)$$

where $l_u(t)$ is called the *rate multiplier* of the logical clock at that time. The nodes can adjust the speed of their logical clocks by modifying their rate multipliers.

For any message, the time that passes from the start of broadcast attempt until the recipient node receives it is referred as *message delay*. In [18], [19], the message delay is modeled as a Gaussian random variable due to the central limit theorem because it is thought to be the addition of numerous independent random processes. It is also shown in [12] that the message delay can be modeled as a Gaussian distributed random variable with %99.8 confidence. Hence, it is assumed that the message delay is a normally distributed random variable with mean zero and variance σ^2.

1. ppm=parts per million, i.e. 10^{-6}.

Algorithm 1 FTSP pseudo-code for node v with a fixed reference node whose node identifier is ROOT.

1: **Initialization**
2: $seq_v \leftarrow 0$
3: start periodic timer with period B
4:
5: □ **Upon receiving** $< L_u, seq_u >$ **such that** $seq_v < seq_u$
6: store (H_v, L_u) pair and recalculate least-squares line
7: $seq_v \leftarrow seq_u$
8:
9: □ **Upon timer timeout**
10: **if** $v = ROOT$ **then** $seq_v \leftarrow seq_v + 1$ **endif**
11: $broadcast < L_v, seq_v >$

4 Slow-Flooding Time Information in WSNs

In this section, we consider the execution of FTSP in order to analyze the effect of waiting times on the synchronization accuracy. The pseudo-code of FTSP with a fixed reference node is given in Algorithm 1. It should be noted that extra controls are omitted and only the general strategy is presented in the pseudo-code. It is also assumed that each node knows the identifier of the reference node.

In FTSP, each node v stores a seq_v variable which stores the largest sequence number received from the reference node. Initially, each node sets its sequence number to zero and starts a periodic timer which will fire at every B seconds (Lines 1-3). Upon receiving an up-to-date synchronization message from any neighbor $u \in \mathcal{N}_v$ (Line 5), v stores the synchronization point $\left(H_v(t'), \hat{L}_u(t')\right)$ as a pair (x, Y) in its *least-squares table (LST)*. This table has a capacity of N pairs to hold the most recent N time information. Node v assumes a *linear relationship* between its hardware clock and the received logical clock values, i.e. the estimates of the reference clock. Least-squares regression is performed by using the stored pairs in order to calculate the *estimated regression line*, i.e. least-squares line, which represents the logical clock function of v (Line 6). Finally, the received sequence number is stored (Line 7).

When a timeout event is generated (Line 9), the sequence number is incremented if that node is the reference node (Line 10). Otherwise, the sequence number remains unchanged. It should be noted that the increment of the sequence number implies that the

reference node is initiating a new flooding round. Each node broadcasts a message which carries the value of its logical clock and its sequence number if it has collected sufficient number of pairs to calculate its first least-squares line (Line 11). The applications running on sensor nodes query *getReferenceTime* interface in order to acquire synchronized notion of time, which returns the value of the logical clock calculated by using the least-squares line, i.e. L_v.

4.1 The Drawbacks of Slow-Flooding By Employing Linear Regression

In this subsection, we summarize our observations on the poor performance of slow-flooding when linear regression is employed. As observed, the crucial step of FTSP is the calculation of the estimated regression line which represents the logical clock. The *slope* of this line can be considered as the estimated speed (i.e. drift) of the reference clock with respect to the hardware clock speed of the corresponding node. It can be shown that the error of this multi-hop drift estimation grows exponentially with the network diameter. The magnitude of this error can be decreased by maintaining a *slope history* for storing previously calculated slope values and by considering the median or the average slope value in this history as the current slope. However, we observed that the synchronization performances of these strategies are unsatisfactory and the exponential growth of the estimation error cannot be eliminated.[2]

Due to the poor performance of the drift estimation, waiting times at each hop until they propagate the flood also amplify the estimation errors. Hence, slow-flooding suffers from the growing error of the estimated reference time at each hop and this situation leads to large synchronization errors especially for the far-away nodes from the reference node. Consequently, it can be concluded that linear regression must not be performed on time information which are originated by a far-away node in slow-flooding based time synchronization.

As an alternative, nodes can be restricted to perform linear regression in order to estimate relative hardware clock speed of their neighbors. Hence, linear regression is performed on time information which are originated by direct neighbors of the current node (one hop away). By flooding the estimated speed as well as the value of the reference clock, network-wide synchronization can be achieved. In order to estimate the speed of the reference node, each node is required to multiply the received estimated reference clock speed with the estimated clock speed of the neighbor from which it received the latest synchronization message. However, we observed that this strategy also exhibits an estimation error which grows exponentially with the network diameter.[3]

In conclusion, the undesired effect of waiting times on the synchronization accuracy can be eliminated if the error of the drift estimation is kept small. Hence, another drift estimation strategy which does not exhibit an exponential error with the network diameter is required.

2. See Section A of Appendix for the details.
3. See Section B of Appendix for the details.

Algorithm 2 FCSA pseudo-code for node v with a fixed reference node whose node identifier is ROOT.

1: **Initialization**
2: clear repository and $\forall u \in \mathcal{N}_v$ set $l_u \leftarrow 1$ and $h_v^u \leftarrow 1$
3: $base_v \leftarrow 0$; $lastUpdate_v \leftarrow 0$;$seq_v \leftarrow 0$
4: $l_v \leftarrow 1$
5: set periodic timer with period B
6:
7: □ **Upon receiving** $< L_u, H_u, l_u, seq_u >$
8: store (H_v, H_u) and estimate h_v^u using least-squares
9: store (h_v^u, l_u) and update l_v
10: **if** $seq_v < seq_u$
11: $base_v \leftarrow L_u$
12: $lastUpdate_v \leftarrow H_v$
13: $seq_v \leftarrow seq_u$
14: **endif**
15
16:□ **Upon timer timeout**
17: **if** $u = ROOT$ **then** $seq_u \leftarrow seq_u + 1$ **endif**
18: **broadcast** $< L_v, H_v, l_v, seq_v >$

5 Slow-Flooding with Clock Speed Agreement

In this section, we describe Flooding With Clock Speed Agreement (FCSA) approach, where all nodes agree on a common logical clock speed by employing a clock speed agreement algorithm and synchronize to a reference node which floods stable time for the whole network. As a consequence, the amplification of estimation errors due to the waiting times is minimized and a synchronization error which grows with the square root of the network diameter is obtained. The pseudo-code of the FCSA algorithm is presented in Algorithm 2.

Any node $v \in V$ executing FCSA maintains a l_v variable, i.e. the rate multiplier, which is adjusted to speed up or slow down the progress of its logical clock. In FCSA, the logical clock value at any time t is calculated as follows:

$$\begin{aligned} L_v(t) &= base_v(t) \\ &\quad + (H_v(t) - lastUpdate_v(t))\, l_v(t). \end{aligned} \tag{3}$$

The variable $base_v$ is required to store the reference clock estimate carried by the most recent synchronization message received and $lastUpdate_v$ holds the hardware clock reading at the time at which $base_v$ variable is updated. It can be noticed from the equation 3 that L_v is calculated by adding up the amount of progress since the latest update of the base value.

Node v maintains a repository in order to keep track of of the relative hardware clock rates and rate multipliers of its neighbors. Due to the memory constraints of the

sensor nodes, the amount of memory dedicated to this repository must be specified in advance. Hence, the maximum number of neighbors of a sensor node and the data stored for each neighbor are limited. When v is powered on, the neighbor repository is cleared, the estimated relative hardware clock rates and rate multipliers kept for each neighbor are initialized to 1 (Line 2). seq_v variable which stores the largest sequence number received from the reference node, $base_v$ and $lastUpdate_v$ variables are initialized to zero (Line 3). The rate multiplier of the current node is set to one (Line 4). Moreover, a periodic timer is started which will fire every time the hardware clock progresses B units (Line 5).

Each time a synchronization message is received from any neighboring node $u \in \mathcal{N}_v$, the (H_v, H_u) pair and the estimated relative hardware clock rate h_v^u is stored in the slot assigned for that neighbor in the repository (Lines 7-8). It should be noted that the slope of the least-squares line which is calculated by using (H_v, H_u) pairs, is an estimate for the relative hardware clock rate h_u/h_v. The main point of FCSA is the Line 9 of the Algorithm 2. Using the values of h_v^u and l_u stored for each $u \in \mathcal{N}_v$, the rate multiplier of v is updated as follows (Line 9):

$$l_v(t^+) \quad = \quad \frac{l_v(t) + \sum_{u \in \mathcal{N}_v} h_v^u(t).l_u(t)}{|\mathcal{N}_v| + 1}. \qquad (4)$$

such that t^+ denotes the time just after the update operation. With this execution, it can be proven theoretically that all nodes agree on a common logical clock speed $h_v.l_v$.[4]

While performing logical clock speed agreement, the estimated clock value of the reference node carried by the message is also considered to update the base value of the logical clock. If the received synchronization message carries a higher sequence number (Line 10), this situation indicates that the reference node has recently flooded its current logical clock value into the network. Hence, v sets the base value of its logical clock to the received logical clock value (Line 11). The hardware clock of v is stored at $lastUpdate_v$ (Line 12) and the sequence number is updated (Line 13). With this execution, each received synchronization message is considered to update the new value of the rate multiplier. However, only the messages which belong to the new synchronization round (which carry higher sequence numbers) are considered to update the value of the logical clock.

When a timeout event is generated (Line 16), only the reference node increments its sequence number and initiates a new flooding round (Line 17). Each node broadcasts a message which carries the value of its logical clock, hardware clock, rate multiplier and sequence number (Line 18).

After the clock speed agreement is established, all nodes run at the same clock speed and the amplification of the estimation errors due to waiting times are eliminated. The only error source is the uncertainties through the message path. As a consequence, we reach the following theorem:

Theorem 5.1: The synchronization error of FCSA grows with the square root of the network diameter.

Proof: See section C of the Appendix for the proof. □

5.1 Synchronization to the Real-Time

If the reference node is equipped with a GPS receiver and has access to real-time, by taking two observations of real-time t_0 and t_1 such that $t_0 < t_1$ after all of the nodes agreed on a common clock rate, it can estimate its hardware clock rate as

$$\hat{h}_r \quad = \quad (H_r(t_1) - H_r(t_0))/(t_1 - t_0). \qquad (5)$$

Using this value, the reference node can also estimate the common logical clock speed $\hat{s} = \hat{h}_r.l_r$ inside the network after the clock speed agreement. By extending the synchronization messages such that they also carry the real-time value t and the estimate of common logical clock speed $\hat{s}$, each node v can estimate the speed of its hardware clock as $\hat{h}_v = \hat{s}/l_v$ since it knows the value of l_v. Node v can also estimate the real-time value at any time using the equation

$$\begin{aligned} rt_v(t) \quad &= \quad rtbase_v(t) \\ &\quad +(H_v(t) - lastUpdate_v(t))/\hat{h}_v(t). \end{aligned} \qquad (6)$$

such that the variable $rtbase_v$ is required to store the value of the real-time value carried by the most recent synchronization message received.

5.2 Reference Node Election

As shown in Algorithm 2, the reference node is predefined before the deployment of the sensor network. Hence, FCSA cannot maintain network-wide synchronization in case of a reference node failure. However, the simple root election mechanism employed in FTSP protocol, i.e. electing the node with the smallest identifier among the remaining nodes as the new reference node, can easily be integrated to FCSA.

5.3 Neighbor Detection And Removal

FCSA can be implemented by employing a simple neighbor detection and removal strategy. Whenever a sensor node receives a synchronization message from a new neighbor, it assigns a free slot for that neighbor and starts collecting its information. If any node does not receive a synchronization message from one of its neighbors for a predefined amount of time, it empties the slot which is assigned for that neighbor in the neighbor repository. As a final point, when a new node joins

4. See Section C of Appendix.

the network, it does not participate in the clock speed agreement immediately. Instead, it first listens a few synchronization packets from its neighbors in order to achieve initial synchronization.

It is infeasible for sensor nodes to keep track of all their neighbors when the number of their neighbors is greater than the capacity of their neighbor repository which is specified in advance. The decision of which neighbors to keep track and which ones to discard is a crucial problem. The communication graph, constructed by considering the neighborhood relations in the neighbor repositories of the sensor nodes, needs to be connected for all times in order to achieve clock speed agreement. One simple solution which prevents the occurrence of this problem is to specify the capacity of the repository as the *maximum node degree* in the network. However, due to the memory constraints, this solution may not work for networks with high neighborhood density.

6 Testbed Experiments

In this section, we present experimental results in order to verify that our theoretical findings are consistent with the practice. We implemented FCSA in TinyOS 2.1.1[5] for our experiments. Since our main attention is flooding based time synchronization, we considered two flooding based time synchronization protocols FTSP and PulseSync for performance comparison. Moreover, we also wanted to make a comparison of FCSA with GTSP since it also employs the same clock speed agreement. We used the publicly available implementation of FTSP in TinyOS. Since PulseSync and GTSP do not have any publicly available implementation, we also implemented them in TinyOS ourselves.

For the evaluation, we considered the instantaneous differences between the logical clocks of the nodes, i.e. *clock skew*. We focused on the largest clock skew between any two nodes, i.e. *global skew,* and between neighboring nodes, i.e. *local skew*. We also took into account the *average global skew* which is defined as the instantaneous average of the global skew and the *average local skew* which is defined as the instantaneous average of the local skew by considering all nodes.

6.1 Hardware Platform

The hardware platform used for the implementation and experiments is MICAz from Memsic[6]. We used 7.37 MHz quartz oscillator on the MICAz board as the clock source for the timer used for getting the local time. The timer operates at 1/8 of that frequency and thus each timer tick occurs at approximately every 921 kHz (approximately 1 microsecond). For MAC layer timestamping, we used CC2420 radio chip implementation of *packet level time synchronization* interfaces [20] provided by TinyOS.

5. http://www.tinyos.net
6. http://www.memsic.com

Table 1: Summary of the experimental measurements on the line topology.

	FTSP	**PulseSync**	**FCSA**	**GTSP**
Max. Global Skew	669 μsec	25 μsec	25 μsec	34 μsec
Avg. Global Skew	538 μsec	18 μsec	19 μsec	26 μsec
Max. Local Skew	519 μsec	8 μsec	16 μsec	10 μsec
Avg. Local Skew	74 μsec	2 μsec	5 μsec	4 μsec

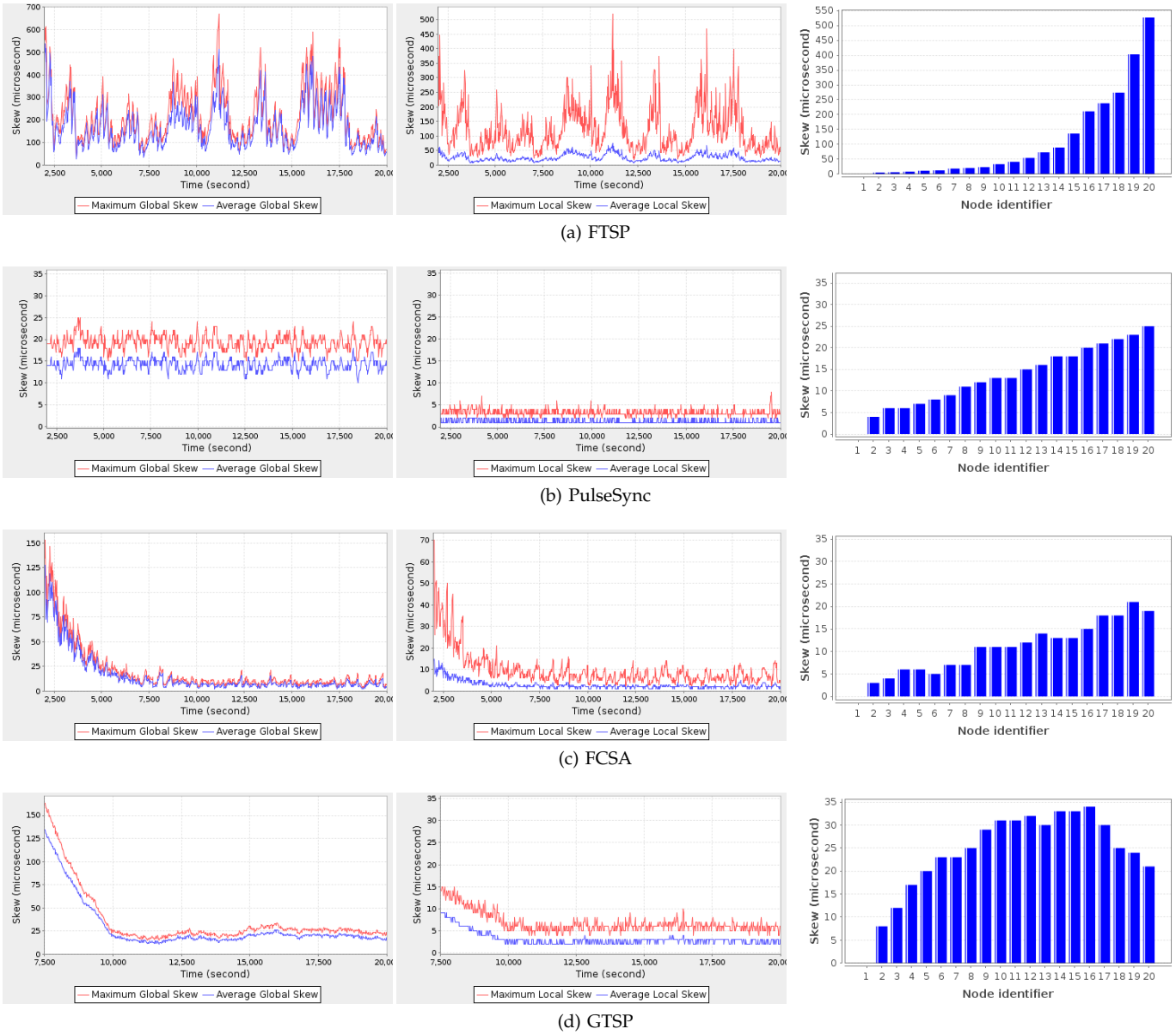
6.2 Testbed Setup

For our experiments, we constructed a line and a 4x5 grid topology of 20 sensor nodes. We have chosen the line topology since the performance of the slow-flooding degrades as the diameter of the network increases and we could get larger diameters with this topology [4]. Grid topology allowed us to compare the performances of slow and rapid flooding approaches when contention and congestion are not negligible as in the line topology. During the experiments each of which took approximately 20000 seconds, the beacon period was 30 seconds and the number of entries for the protocol tables, i.e. regression table and neighbor repository, was 8. The environmental conditions were quite stable and there were not sudden temperature changes. We powered on sensor nodes randomly in the first 3 minutes. We collected clock values from all sensor nodes at the end of each interval which is uniformly distributed between 20 and 23 seconds during the experiments. At the beginning of each interval, the reference broadcaster node transmits a packet. This message is received approximately at the same time by all nodes. Then, the nodes broadcast their clock values and rate multipliers at the receipt time of this packet. The base station node attached to a PC transfers these messages to the serial port. An application listening the serial port logs these messages. Hence, the logical clock values and rate multipliers of the nodes at specific time instants have been collected during the experiments. At the end of the experiments, the evaluation metrics are applied to the collected data and the results are analyzed.

6.3 Experimental Results

Figures 1 and 2 show the global and local skew values, the skew values to the reference node and the slope values of the estimated regression line (or alternatively rate multipliers) during the experiments on the line topology with FTSP, PulseSync, FCSA and GTSP, respectively. Table 1 summarizes the skew values observed during these experiments. It should be noted that node 1 is the time reference for the other nodes in the network for FTSP, PulseSync and FCSA.

It can be observed that as the distance from the node 1 gets larger, the collected slope values show quite variability and they are unstable for FTSP. If node with identifier 20 is considered, the amplitude of the variation of its slope values is much higher than that of the slope



(a) FTSP

(b) PulseSync

(c) FCSA

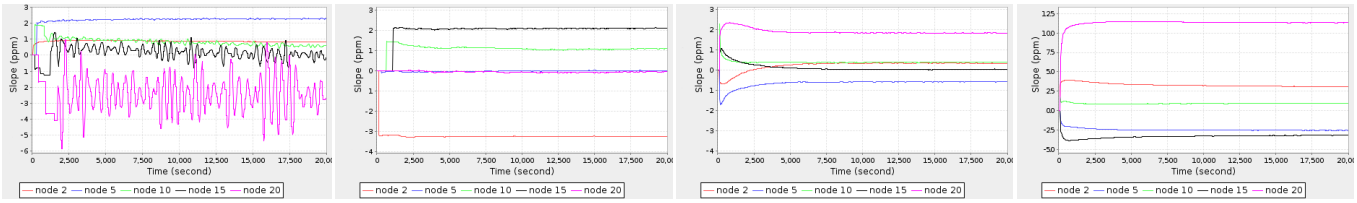
(d) GTSP

Figure 1: Global skew (left column), local skew (middle column) and the maximum synchronization error versus distance from the reference (root) node 1 (right column) on the line topology for FTSP, PulseSync, FCSA and GTSP, respectively.

values of the other nodes. This poor performance of the drift estimation has lead to large skew values especially for the far-away nodes from the reference node. Hence, FTSP exhibited quite large global and local skew values.

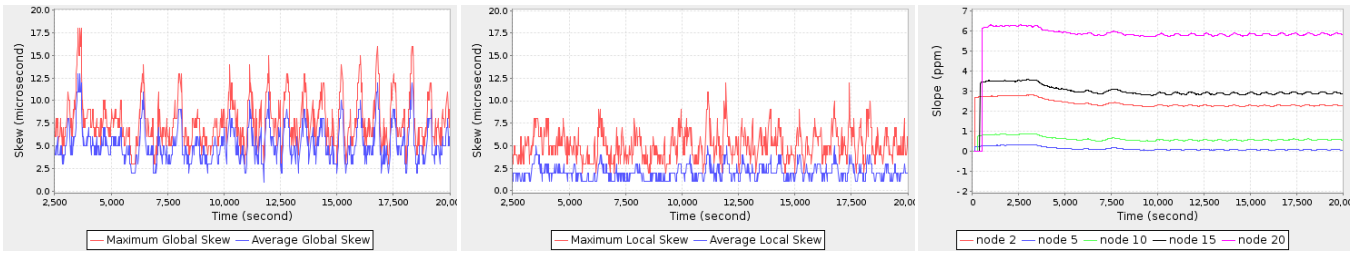
It can be concluded that PulseSync reduced the error of the drift estimation considerably when compared to FTSP since it employs a rapid flooding approach and reduces the amplification of the estimation errors at each hop. However, far-away nodes from the reference node still suffer from relatively higher slope variations when compared to the nearby nodes to the reference. But these variations are dramatically smaller than that in FTSP. The rapid flooding strategy reduced the estimation errors considerably. Hence, PulseSync achieved quite tight synchronization in terms of local and global skew during the experiments.

As can be observed, the negative effect of slow-flooding on the scalability is reduced considerably with the clock speed agreement employed by FCSA. Since the rate multipliers are quite stable, the estimation errors of the far-away nodes are small when compared to FTSP. It can be concluded that FCSA outperforms FTSP quite significantly and catches the performance of PulseSync although it does not change the propagation speed of the flood, employs an identical message pattern and has the same message complexity. If maximum global skew is considered, FCSA improved the performance of FTSP approximately by a factor of 26 and performed nearly the same when compared to PulseSync. On the other hand, the local skew values observed with PulseSync were smaller than that of with FCSA, as expected. The reason is that nodes get the

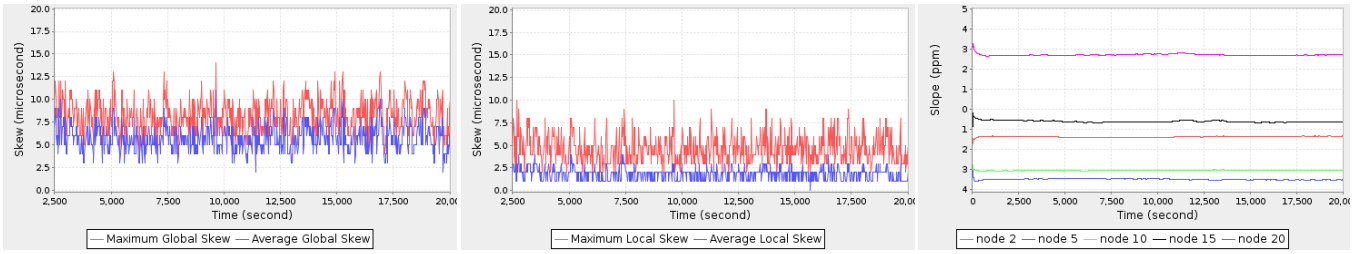


(a) FTSP (b) PulseSync (c) FCSA (d) GTSP

Figure 2: The slope of the estimated regression line (from which 1.0 is subtracted) on the line topology for FTSP and PulseSync, rate multipliers (from which 1.0 is subtracted) for FCSA and GTSP, respectively.



(a) PulseSync



(b) FCSA

Figure 3: Global skew (left column), local skew (middle column) and the slope of the estimated regression line/rate multipliers from which 1.0 is subtracted (right column) on the 5x4 grid topology for PulseSync and FCSA.

recent time information of the reference node and update themselves much more quickly with rapid-flooding and this eliminates the occurrence of large instant local skews when compared to slow-flooding.

With FCSA, the initial network wide synchronization was achieved approximately at the 400th second, which took shorter than that with FTSP, which was established approximately at the 2000th second. The global skew of FCSA was 390 μsec at that time. The actual performance of FCSA can be observed as soon as the clock speed agreement is established. In the experiments, tight synchronization was achieved approximately at the 5000th second. Although FCSA requires longer time for tight synchronization, the synchronization error was still within acceptable boundaries, i.e. between $390\mu s$ and $30\mu s$, in the interval [400-5000] until agreement is achieved.

The main objective of GTSP is to provide tight synchronization between neighboring nodes by employing a clock speed and then a clock value agreement (by considering the clock values of their neighboring nodes) among the neighboring sensor nodes. However, this two-phase agreement increased the time the agreement took (agreement was established at approximately 10000th second of the experiments)

Table 2: Summary of the experimental measurements on the 5x4 grid topology.

	PulseSync	**FCSA**
Max. Global Skew	18 μsec	14 μsec
Avg. Global Skew	13 μsec	11 μsec
Max. Local Skew	12 μsec	10 μsec
Avg. Local Skew	5 μsec	4 μsec

and we observed increased global skew with this mechanism when compared to FCSA. Since there is not any reference node and any flooding mechanism, the synchronization errors of far-away and nearby nodes to the node 1 are quite similar in GTSP, as can be observed from the Figure 1. Moreover, the local skew of GTSP is smaller than that of FCSA since FCSA only employs clock speed agreement and nodes agree on the clock value of the reference node by considering the flooded time information of this node, not considering the time information of their neighboring nodes. However, this mechanism allows FCSA to be used for external synchronization, which makes it superior than GTSP.

In order to clarify the shortcomings of the fast-flooding approach, we also performed experiments on the 5x4

grid topology and hence compared the performances of PulseSync and FCSA when the neighborhood density is increased. Figure 3 shows the experimental results and Table 2 presents the summary of the observed skew values on this topology. PulseSync requires that time information must be propagated quickly and reliably through the network with the pulses. However, due to neighborhood contention and increased rate of the packet losses, rapid-flooding on the grid topology is difficult when compared to that on the line topology. It can be observed that the global and local skews of FCSA is superior than those of PulseSync on the grid topology. The rate multipliers of the nodes executing FCSA are quite stable when compared to those of the nodes of executing PulseSync. These results suggest that the slow-flooding based FCSA is superior when contention and congestion are not negligible in a sensor network.

As a summary of our experiments, it can be concluded that FCSA achieves the tight synchronization quality of rapid-flooding by employing slow-flooding and clock speed agreement together although it does not change the communication frequency.

7 Simulations

Our experimental results collected from a small network showed that the PulseSync and FCSA exhibit quite comparable synchronization errors and GTSP is the worst among them in terms of synchronization accuracy. We wanted to observe if this situation also holds on networks with larger diameters. Hence, in addition to real-world experiments, we implemented FCSA, GTSP and PulseSync in our WSN simulator which we implemented with Java programming language to compare their performances on longer networks. During our simulations, we implemented the hardware clocks of nodes in software with a drift which is uniformly distributed between ± 50 ppm. We modeled the variances in the message delay with a normally distributed random variable. We applied our evaluation metrics for line topology with different diameters. For each diameter, we performed 10 simulation runs and averaged the calculated skews for these runs. Figure 4 presents the maximum synchronization error observed during the simulations. It can be concluded that the performances of FCSA and PulseSync are also quite comparable and their synchronization error grow quite similarly and substantially slowly with the network diameter. As in the real-world experiments, GTSP exhibited the worst synchronization performance during the simulations.

In order to observe the behavior of FCSA when the neighborhood density is increased, we also performed simulations on grid and line topologies with the same diameters. Table 3 presents a summary of these simulations. It can be concluded that as the number of neighbors of sensor nodes increases, the time required

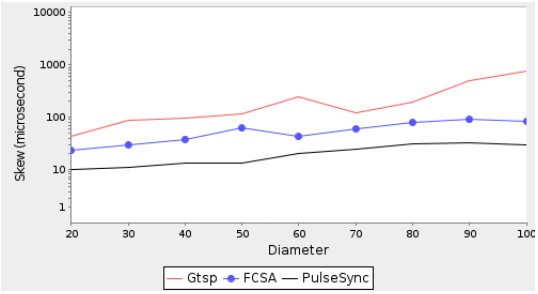


Figure 4: Simulation results on the line topology with different diameters.

Table 3: Simulation results of FCSA on grid and line topologies.

	Agreement Time	Max. Global
4x4 Grid vs Line 6	750 s / 750 s	16 μsec / 15 μsec
8x8 Grid vs Line 14	2200 s / 4500 s	26 μsec / 30 μsec
16x16 Grid vs Line 30	8500 s / 12000 s	49 μsec / 48 μsec
32x32 Grid vs Line 62	22000 s / 108000 s	64 μsec / 94 μsec

until all nodes agree on a common clock speed decreases. On the line topology, the messages originated from the reference node follow the same path to reach the node at the end of the network. On the contrary to the line topology, the synchronization messages of the reference node may follow many paths in order to reach to the far-away nodes on the grid topology and far-away nodes may collect time information with smaller error. We observed quite similar skew values with grid and line topologies for diameters up to 30. However, we observed smaller synchronization error with the grid topology when the diameter is 62.

We also performed simulations in order to evaluate the adaptation of FCSA and PulseSync to the changing environmental conditions. Please see Appendix D for the details. We conclude from our simulations that the adaptation time of FCSA is not too long when compared to PulseSync. Hence, FCSA is also desirable when environmental conditions are prone to change frequently.

8 Conclusion and Future Work

In this study, we considered the question of whether it is possible to achieve scalable and tight synchronization in WSNs with slow flooding. We emphasized that the performance of drift estimation mechanism is vital for flooding based time synchronization protocols. We pointed out that waiting times due to slow-flooding with multi-hop least-squares drift estimation amplifies the estimation errors at each hop. Hence, we revealed that the smaller the error of the drift estimation and hence the difference between the speeds of the clocks, the smaller the undesired effect of waiting times on the synchronization accuracy. As a main contribution of this paper, we showed that the synchronization quality of slow-flooding based time synchronization

can drastically be improved by employing a clock speed agreement algorithm among the sensor nodes. We introduced Flooding With Clock Speed Agreement (FCSA) protocol which forces all nodes to run at the same speed by employing an agreement algorithm and synchronizes them to a reference node which floods stable time for the whole network. We presented the synchronization accuracy of this protocol in our experimental testbed and compared its performance to FTSP protocol which also employs slow-flooding, to PulseSync protocol which employs rapid-flooding approach and to GTSP which employs clock speed agreement but not flooding. It can be concluded from our experiments on the line topology that FCSA improves the performance of GTSP remarkably and outperforms FTSP by a factor of 26. Moreover, FCSA catches the performance of PulseSync on the line topology and it is superior on the grid topology where congestion and contention are not negligible. Our simulations showed that these conclusions also hold on larger and denser networks.

Although FCSA decreased the time required to achieve initial network-wide synchronization, it requires a long time until quite tight synchronization is achieved. For the line topology of 20 sensor nodes, it took approximately 5000 seconds (approximately 1.5 hours) for nodes to establish clock speed agreement to achieve tight synchronization. We leave speeding up the agreement time as a future work. Secondly, the clock speed agreement may not be achieved in networks with high neighborhood density, since the network graph which is constructed by considering the neighbor repositories of the sensor nodes may loss its connectivity due to the limited capacity of these repositories. We leave devising a solution to the problem of deciding which neighbors to keep track and which neighbors to discard while preserving connectivity as an important future work. Finally, we plan to evaluate the performance of FCSA in synchronization to the real time.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their constructive comments to improve this manuscript. Kasım Sinan YILDIRIM acknowledges The Turkish Scientific and Technical Research Council (TÜBİTAK) for supporting this work through a domestic PhD scholarship program (BAYG-2211).

REFERENCES

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

[2] I. F. Akyildiz and M. C. Vuran, *Wireless Sensor Networks*. John Wiley & Sons, 2010.

[3] M. Maróti, B. Kusy, G. Simon, and A. Lédeczi, "The flooding time synchronization protocol," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*. New York, NY, USA: ACM, 2004, pp. 39–49.

[4] C. Lenzen, P. Sommer, and R. Wattenhofer, "Optimal Clock Synchronization in Networks," in *7th ACM Conference on Embedded Networked Sensor Systems (SenSys), Berkeley, California, USA*, November 2009.

[5] J. Lu and K. Whitehouse, "Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks," in *INFOCOM 2009, IEEE*, april 2009, pp. 2491 –2499.

[6] T. Schmid, Z. Charbiwala, Z. Anagnostopoulou, M. B. Srivastava, and P. Dutta, "A case against routing-integrated time synchronization," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems*, ser. SenSys '10. New York, NY, USA: ACM, 2010, pp. 267–280.

[7] R. Gotzhein and T. Kuhn, "Black burst synchronization (bbs) – a protocol for deterministic tick and time synchronization in wireless networks," *Computer Networks*, vol. 55, no. 13, pp. 3015 – 3031, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128611001812

[8] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with glossy," in *Information Processing in Sensor Networks (IPSN), 2011 10th International Conference on*, april 2011, pp. 73 –84.

[9] P. Sommer and R. Wattenhofer, "Gradient Clock Synchronization in Wireless Sensor Networks," in *8th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), San Francisco, USA*, April 2009.

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

[11] T. Schmid, Z. Charbiwala, R. Shea, and M. Srivastava, "Temperature compensated time synchronization," *Embedded Systems Letters, IEEE*, vol. 1, no. 2, pp. 37 –41, aug. 2009.

[12] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Oper. Syst. Rev.*, vol. 36, no. SI, pp. 147–163, 2002.

[13] J. van Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2003, pp. 11–19.

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

[15] H. Dai and R. Han, "Tsync: A lightweight bidirectional time synchronization service for wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, pp. 125–139, 2004.

[16] K. Sun, P. Ning, and C. Wang, "Tinysersync: secure and resilient time synchronization in wireless sensor networks," in *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2006, pp. 264–277.

[17] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler, "Elapsed time on arrival: A simple and versatile primitive for canonical time synchronisation services," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 1, no. 4, pp. 239–251, 2006.

[18] K. lae Noh, E. Serpedin, and K. Qaraqe, "A new approach for time synchronization in wireless sensor networks: Pairwise broadcast synchronization," *Wireless Communications, IEEE Transactions on*, vol. 7, no. 9, pp. 3318 –3322, 2008.

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

[20] M. Maroti and J. Sallai, "Packet-level time synchronization," TinyOS Core Working Group, Technical Report, May 2008.

[21] S. M. Ross, *Introduction to Probability and Statistics for Engineers and Scientists, Third Edition*. Elsevier Academic Press, 2004.

[22] E. J. Dietz, "A comparison of robust estimators in simple linear regression," vol. 16, pp. 1209–1227, 1987.

[23] R. R. Wilcox, *Fundamentals of Modern Statistical Methods: Substantially Improving Power and Accuracy*, 2nd ed. Springer New York, 2010.

[24] G. Xiong and S. Kishore, "Analysis of distributed consensus time synchronization with gaussian delay over wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2009, no. 1, p. 528161, 2009.

Kasım Sinan Yıldırım received the B.Sc., M.Sc. and Ph.D. degrees in computer engineering from Ege University, İzmir, Turkey in 2003, 2006 and 2012 respectively. He works as a research assistant at the Department of Computer Engineering at Ege University since 2007. His research interests are embedded systems, distributed systems, distributed algorithms and wireless sensor networks.

Aylin Kantarcı received the B.Sc., M.Sc. and Ph.D. degrees from Ege University, İzmir, Turkey, in 1992, 1994 and 2000, respectively. She is an associate professor at the Department of Computer Engineering at Ege University. Her current research issues include distributed systems, wireless sensor networks, and multimedia.

APPENDIX A
SYNCHRONIZATION ERROR OF SLOW-FLOODING WITH LEAST-SQUARES REGRESSION

In least-squares regression, the linear relationship between the collected $\{(x_i, Y_i) \in LST | i = 0, .., N-1\}$ pairs is modeled as $Y_i = \alpha + \beta x_i + \varepsilon_i$, such that $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$. If we let $\overline{x} = \sum x_i/N$, $\overline{Y} = \sum Y_i/N$, $S_{xx} = \sum (x_i - \overline{x})^2$ and $S_{xY} = \sum (x_i - \overline{x}) Y_i$, the *slope* $\hat{\beta}$ and the *intercept* $\hat{\alpha}$ of the least-squares line $\hat{Y} = \hat{\alpha} + \hat{\beta} x$ are computed as [21]:

$$\hat{\beta} = S_{xY}/S_{xx}, \quad (7)$$

$$\hat{\alpha} = \overline{Y} - \hat{\beta}\overline{x}. \quad (8)$$

Upon receiving a new synchronization message, the earliest pair is removed from LST, the received pair is stored in LST and the least-squares line is recalculated. Hence, the intercept and the slope of the least-squares line may change at each synchronization message receipt. By using the recently calculated least-squares line, any node v can predict Y values which will be received in the future. The distribution of $Y_{x'} - \hat{Y}_{x'}$, which represents the error between the predicted and the actual Y value at a single future point $x^{'}$ is given as follows [21]:

$$Y_{x'} - \hat{Y}_{x'} \sim \mathcal{N}\left(0, \sigma^2 \left(1 + \frac{1}{N} + \frac{(x^{'} - \overline{x})^2}{S_{xx}}\right)\right). \quad (9)$$

Hence, the larger the difference between $x^{'} = H_v(t)$ and $\overline{x}$, the larger the error of the predicted reference clock value $L_v(t)$. It should be noted that until a recent (x, Y) pair is collected and the new least-squares line is recalculated, the value of $H_v(t) - \overline{x}$ increases as time passes. Whenever a recent (x, Y) pair is collected, the value of $H_v(t) - \overline{x}$ decreases upon the recalculation of the least-squares line. This situation leads us to the following fact.

Fact 1: From the time at which any node $v \in V$ calculated its current least-squares line to the time at which the new least-squares line will be calculated upon receiving a new synchronization point, the variance of the estimated reference clock values increases. Hence, slow-flooding suffers from the growing error of the estimated reference time due to the waiting times at each sensor node until they propagate the flood.

We now present an analysis of FTSP by following the analysis presented in [4], [6]. Since the analysis becomes more complex for the larger sizes of LST, the size of the LST is considered as $N = 2$ for simplicity. For this case, the intercept $\hat{\alpha}$ and the slope $\hat{\beta}$ of the *initial* estimated regression line are calculated by considering only two pairs (x_0, y_0) and (x_1, y_1) in LST as follows:

$$\hat{\beta} = \frac{\sum_{i=0}^{1} (x_i - \overline{x}) y_i}{\sum_{i=0}^{1} (x_i - \overline{x})^2} = \frac{y_1 - y_0}{x_1 - x_0}, \quad (10)$$

$$\hat{\alpha} = \frac{y_1 + y_0}{2} - \hat{\beta}\left(\frac{x_1 + x_0}{2}\right). \quad (11)$$

Upon receiving a new message, the earliest pair is removed from LST and the recently received pair is stored. Then, the intercept and the slope of the estimated regression line are re-calculated by considering the pairs in LST.

Now, consider the execution of FTSP on a line of n nodes $v_o, v_1, ..., v_{n-1} \in V$ such that v_0 is the reference node. During the execution, v_1 collects synchronization messages from the reference node v_0 which carry *independent* hardware clock readings. v_1 stores the received time information as (x_{1i}, y_{0i}) such that y_{0i} represents the ith received time information from v_0 and x_{1i} represents the corresponding hardware clock value of v_1. Each received y_{0i} suffers from an independent error $\varepsilon_{0i} \sim \mathcal{N}(0, \sigma^2)$ which occurred due to the message delays. After v_1 has collected two messages, it establishes a linear relationship between its hardware clock and the hardware clock of the reference node as $y_{0i} = \alpha + \beta x_{1i} + \varepsilon_{0i}$. In order to estimate the real intercept α and the real slope β parameters, v_1 performs least-squares regression. The variance of the estimated slope $\hat{\beta}_{1i}$ which is calculated by considering the pairs (x_{1i}, y_{0i}) and $(x_{1(i+1)}, y_{0(i+1)})$ can be obtained as follows:

$$Var(\hat{\beta}_{1i}) = Var\left(\frac{y_{0(i+1)} - y_{0i}}{x_{1(i+1)} - x_{1i}}\right) = \frac{2\sigma^2}{(x_{1(i+1)} - x_{1i})^2}. \quad (12)$$

Let $\triangle_{1i}$ denotes the difference of hardware clock readings of v_1 at the receipt time of the *(i+1)th* and *ith* synchronization message from v_0, i.e. $\triangle_{1i} = x_{1(i+1)} - x_{1i}$. For simplicity, assume that $\triangle_{1i} = \triangle_1$ holds for all i. Then, the variance of $\hat{\beta}_{1i}$ can be written as follows:

$$Var(\hat{\beta}_{1i}) = \frac{2\sigma^2}{\triangle_1^2}. \quad (13)$$

Nodes can estimate future hardware clock values of the reference node by using their estimated regression lines and hardware clock readings. Upon receiving a synchronization message and updating the estimated regression line, each node waits an amount of time and broadcasts its estimate about the hardware clock of the reference node in order to achieve network-wide synchronization. Let $\dot{x}_{1i}$ be the hardware clock value at which v_1 broadcasted its ith estimate y_{1i}. By using the estimated regression line, this estimate is calculated as follows:

$$y_{1i} = \frac{y_{0(i+1)}+y_{0i}}{2} + \hat{\beta}_{1i}(\dot{x}_{1i} - \frac{x_{1(i+1)}+x_{1i}}{2}). \quad (14)$$

Assume that $\dot{x}_{1i} > x_{1(i+1)} + \frac{x_{1(i+1)}-x_{1i}}{2}$ holds. This means that more than $\frac{x_{1(i+1)}-x_{1i}}{2}$ hardware clock ticks are waited after the message receipt at local time $x_{1(i+1)}$ in order to broadcast the estimate y_{1i}. We assume that *each node waits approximately the same amount of local time* (hardware clock progress) in order to propagate its estimate to its neighboring nodes upon a message receipt. Hence, we assume that $\dot{x}_{1i} - \frac{x_{1(i+1)}+x_{1i}}{2} = W > \triangle_1$. If we denote $\frac{y_{0(i+1)}+y_{0i}}{2}$ by $\overline{y}_{0i}$, the equality above can be written as follows:

$$y_{1i} = \overline{y}_{0i} + \hat{\beta}_{1i}W. \quad (15)$$

In order to calculate the variance of this estimate, we are required to calculate $Var(\overline{y}_{0i})$ and $cov(\overline{y}_{0i}, \hat{\beta}_{1i})$. These calculations are performed by applying the following steps:

$$Var(\overline{y}_{0i}) = Var(\frac{y_{0(i+1)}+y_{0i}}{2}) = \frac{\sigma^2}{2}. \quad (16)$$

$$\begin{aligned} Cov(\overline{y}_{0i}, \hat{\beta}_{1i}) &= Cov(\frac{y_{0(i+1)}+y_{0i}}{2}, \frac{y_{0(i+1)}-y_{0i}}{x_{1(i+1)}-x_{1i}}) \\ &= \frac{Var(y_{0(i+1)}) - Var(y_{0i})}{2(x_{1(i+1)}-x_{1i})} = 0. \end{aligned} \quad (17)$$

The last step holds since $y_{0(i+1)}$ and y_{0i} are independent, hence $cov(y_{0(i+1)}, y_{0i}) = 0$. By using the equalities above, the variance of y_{1i} is calculated as follows:

$$\begin{aligned} Var(y_{1i}) &= Var(\overline{y}_{0i}) + W^2 Var(\hat{\beta}_{1i}) \\ &\quad +2WCov(\overline{y}_{0i}, \hat{\beta}_{1i})) \\ &= \sigma^2 \left(\frac{1}{2} + \frac{2W^2}{\Delta_1^2}\right). \end{aligned} \quad (18)$$

In the following steps of our analysis, we require the covariance of two estimates $y_{1(i+1)}$ and y_{1i} which are broadcasted successively by node v_1. This covariance is obtained as follows:

$$\begin{aligned} Cov(y_{1(i+1)}, y_{1i}) &= Cov(\overline{y}_{0(i+1)} + \hat{\beta}_{1(i+1)}W, \overline{y}_{0i} + \hat{\beta}_{1i}W) \\ &= Cov(\overline{y}_{0(i+1)}, \overline{y}_{0i}) \\ &\quad +WCov(\overline{y}_{0(i+1)}, \hat{\beta}_{1i}) \\ &\quad +WCov(\overline{y}_{0i}, \hat{\beta}_{1(i+1)}) \\ &\quad +W^2 Cov(\hat{\beta}_{1(i+1)}, \hat{\beta}_{1i}) \\ &= Var(y_{0(i+1)})(\frac{1}{4} - \frac{W^2}{\Delta_1^2}) \\ &= \sigma^2 \left(\frac{1}{4} - \frac{W^2}{\Delta_1^2}\right). \end{aligned} \quad (19)$$

It should be noted that the covariance of two nonsuccessive estimates is zero since their calculations do not include any common term. For instance, y_{1i} is calculated by considering y_{0i} and $y_{0(i+1)}$ where $y_{1(i+2)}$ is calculated by considering $y_{0(i+3)}$ and $y_{0(i+2)}$.

Now, we focus on node v_2 which collects synchronization messages from v_1 carrying estimates y_{1i}. Apart from their estimation errors, these estimates suffer from an additional independent error $\varepsilon_{1i} \sim \mathcal{N}(0, \sigma^2)$ occurred due to the message delays. After v_2 has collected two messages, it establishes a linear relationship between its hardware clock and the hardware clock of the reference node as $y_{1i} = \alpha + \beta x_{2i} + \varepsilon_{1i}$. After performing least-squares regression, the variance of the estimated slope $\hat{\beta}_{2i}$ which is calculated by considering the pairs (x_{2i}, y_{1i}) and $(x_{2(i+1)}, y_{1(i+1)})$ can be obtained as follows:

$$\begin{aligned} Var(\hat{\beta}_{2i}) &= Var(\frac{y_{1(i+1)}-y_{1i}}{x_{2(i+1)}-x_{2i}}) \\ &= \frac{1}{\Delta_2^2}(Var(y_{1(i+1)}) + Var(y_{1i}) \\ &\quad -2Cov(y_{1(i+1)}, y_{1i}) + 2\sigma^2) \\ &= \frac{\sigma^2}{\Delta_2^2}\left(\frac{5}{2} + \frac{6W^2}{\Delta_1^2}\right). \end{aligned} \quad (20)$$

If it is assumed that $W > \triangle_i$ for all i, the variance of $\hat{\beta}_{2i}$ becomes approximately **a factor of $\frac{W^2}{\Delta_2^2}$ greater** than the variance of $\hat{\beta}_{1i}$.

v_2 broadcasts its estimates about the future hardware clock values of the reference node by using its estimated regression line and its hardware clock reading. The ith broadcasted estimate of v_2 is given as follows:

$$y_{2i} = \overline{y}_{1i} + \hat{\beta}_{2i}W. \quad (21)$$

In order to calculate the variance of this estimate, we are required to calculate $Var(\overline{y}_{1i})$ and $cov(\overline{y}_{1i}, \hat{\beta}_{2i})$, which we obtain by applying the following steps:

$$\begin{aligned} Var(\overline{y}_{1i}) &= Var(\frac{y_{1(i+1)}+y_{1i}}{2}) \\ &= \frac{1}{4}(Var(y_{1(i+1)}) + Var(y_{1i}) \\ &\quad +2Cov(y_{1(i+1)}, y_{1i})) + 2\sigma^2) \\ &= \sigma^2 \left(\frac{7}{8} + \frac{W^2}{2\Delta_1^2}\right). \end{aligned} \quad (22)$$

$$\begin{aligned} Cov(\overline{y}_{1i}, \hat{\beta}_{2i}) &= Cov(\frac{y_{1(i+1)}+y_{1i}}{2}, \frac{y_{1(i+1)}-y_{1i}}{x_{2(i+1)}-x_{2i}}) \\ &= \frac{Var(y_{1(i+1)}) - Var(y_{1i})}{2(x_{2(i+1)}-x_{2i})} = 0. \end{aligned} \quad (23)$$

By using these results, we get the variance of y_{2i} as follows:

$$Var(y_{2i}) = Var(\overline{y}_{1i}) + W^2 Var(\hat{\beta}_{2i}) = \sigma^2 \left(\frac{7}{8} + \frac{2W^2}{\Delta_1^2} + \frac{5W^2}{2\Delta_2^2} + \frac{6W^4}{\Delta_1^2 \Delta_2^2} \right). \quad (24)$$

Finally, the covariance of two successive estimates $y_{2(i+1)}$ and y_{2i} of node v_2 can be obtained as follows:

$$Cov(y_{2(i+1)}, y_{2i}) = (\frac{1}{4} - \frac{W^2}{\Delta_2^2}) Var(y_{1(i+1)}) = \sigma^2 \left(\frac{1}{4} - \frac{W^2}{\Delta_2^2} \right) \left(\frac{1}{2} + \frac{2W^2}{\Delta_1^2} \right) = \sigma^2 \left(\frac{1}{8} + \frac{W^2}{2\Delta_1^2} - \frac{W^2}{2\Delta_2^2} - \frac{2W^4}{\Delta_1^2 \Delta_2^2} \right) \quad (25)$$

By using these results, the variance of the slope of the estimated regression line $\hat{\beta}_{3i}$ of v_3 can be given as follows:

$$Var(\hat{\beta}_{3i}) = Var(\frac{y_{2(i+1)} - y_{2i}}{x_{3(i+1)} - x_{3i}}) = \frac{1}{\Delta_3^2}(Var(y_{2(i+1)}) + Var(y_{2i}) - 2Cov(y_{2(i+1)}, y_{2i}) + 2\sigma^2) = \frac{\sigma^2}{\Delta_3^2} \left(\frac{7}{2} + \frac{3W^2}{\Delta_1^2} + \frac{6W^2}{\Delta_2^2} + \frac{16W^4}{\Delta_1^2 \Delta_2^2} \right). \quad (26)$$

It can be observed that the variance of $\hat{\beta}_{3i}$ is approximately **a factor of** $\frac{W^4}{\Delta_3^2 \Delta_2^2}$ **greater** than the variance of $\hat{\beta}_{1i}$. With this execution, the variance $\hat{\beta}_{(n-1)i}$ of the slope of the estimated regression line calculated by v_{n-1} becomes a factor of $\prod_{j=2}^{n-1} \frac{W^2}{\Delta_j^2}$ greater than the variance of $\hat{\beta}_{1i}$. If $1 < \mathcal{F} \leq \frac{W^2}{\Delta_i^2}$ holds for all i, we reach to the following facts.

Fact 2: During the execution of slow-flooding, the variance of the estimated relative clock rate is amplified at each hop $v_i \in V$ approximately by a factor of $\mathcal{F}$.

Fact 3: As the distance from the reference node increases, far-away nodes collect time information with larger deviations. Thus, their estimation errors are larger than those of the closer nodes to the reference.

Theorem A.1: The variance of the relative drift estimation of node $v_k \in V$ is on the order of $\mathcal{O}(\mathcal{F}^{k-1})$.

It should be noted that the exponential behavior of this multi-hop drift estimation with larger sizes of the LST table has been observed by simulations in [4].

APPENDIX B
IMPROVING THE PERFORMANCE OF SLOW-FLOODING

As stated in the previous section, the variance of the drift estimation with slow flooding is amplified by approximately a factor of $\mathcal{F}$ at each hop. Since nodes do not quickly propagate the flood, slow-flooding suffers from the large values of $\mathcal{F}$, which leads to large synchronization errors. In this section, we will propose new methods which reduce the amplification of errors at each hop and improves the performance of slow-flooding without changing the propagation speed of the flood and its message complexity. In subsection B.1, we propose three methods in order to decrease the variance of the slope of the least-squares line. In subsection B.2, we propose another new method which floods the estimated relative rate and the estimated value of the reference clock together, which reduces the effect of waiting times on the synchronization accuracy considerably.

B.1 Slow-Flooding By Using a Slope History

The probability distribution of the estimates which are calculated by using the least squares line depends on the probability distribution of the slope of this line, which is given as follows [21]:

$$\hat{\beta} \sim \mathcal{N}\left(\beta, \sigma^2 / S_{xx}\right). \quad (27)$$

If the variance of this probability distribution is decreased, the estimation errors will also be decreased. Within this context, we propose three methods which require each node to maintain a *slope history* ($\mathcal{SH}$) for holding recently calculated N slope values of the least-squares line. Hence, it is required that whenever any node recalculates the parameters of its least-squares line (Line 6 of Algorithm 1), it removes the earliest calculated slope value from $\mathcal{SH}$ and stores the recently calculated slope value in $\mathcal{SH}$. Our methods modify *getReferenceTime* interface in such a way that they consider the values in $\mathcal{SH}$ when calculating synchronized notion of time for the applications, rather than returning the value of the logical clock which is calculated by using the current least-squares line.

B.1.1 Considering the Median Slope in $\mathcal{SH}$ (MS)

In least-squares, even a single synchronization point with a large error may distort the slope of the estimated regression line. It is well known that median based methods lead to robust regression lines [22], [23]. By using this reality, we propose considering the median slope in $\mathcal{SH}$ when returning synchronized notion of time to the applications. Hence, *getReferenceTime* interface on any node v is modified to return the value of $\hat{\alpha}_{med} + \hat{\beta}_{med} H_v(t)$ at any time t such that $\hat{\beta}_{med} = median\{\beta_i \in \mathcal{SH}, |i = 0..N-1\}$ and $\hat{\alpha}_{med} = \overline{Y} - \hat{\beta}_{med}\overline{x}$. We claim that the variance of $\hat{\beta}_{med}$ is smaller than that of $\hat{\beta}$, since $\hat{\beta}_{med}$ is more robust to points with large errors. A disadvantage of MS is that it considers only the median slope as the slope of the estimated regression line. Hence, other slope values in $\mathcal{SH}$ have *zero weight* during the calculation of $\hat{\beta}_{med}$.

B.1.2 Considering the Average Slope in $\mathcal{SH}$ (AS)

Similar to the median based approach, we now propose considering the average slope in $\mathcal{SH}$ when returning synchronized notion of time to the applications. Hence, *getReferenceTime* interface on any node v is modified to return the value of $\hat{\alpha}_{avg} + \hat{\beta}_{avg} H_v(t)$ at any time t such that $\hat{\beta}_{avg} = \sum_{\beta_i \in \mathcal{SH}} \beta_i / N$ and $\hat{\alpha}_{avg} = \overline{Y} - \hat{\beta}_{avg}\overline{x}$. It should be noted that all of the slope values in $\mathcal{SH}$ have *equal weight* during the calculation of $\hat{\beta}_{avg}$. Since each slope in $\mathcal{SH}$ is an independent estimate of the slope of the real regression line, the variance of $\hat{\beta}_{avg}$ is smaller than that of $\hat{\beta}$ by a factor of $1/N$.

B.1.3 Considering the Average Slope in $\mathcal{SH}$ Together With the Slope of the Recently Calculated Least-Squares Line (ACS)

When the values in $\mathcal{SH}$ have large errors, taking their average or median may reduce the estimation errors. However, when a more accurate slope value is obtained by the recently calculated least-squares line, previous slope values with large errors in $\mathcal{SH}$ may diminish the contribution of this value. In order to prevent this situation, we now propose taking the average of the average slope in $\mathcal{SH}$ and the slope of the current least-squares line. The value of $\hat{\alpha}_{acs} + \hat{\beta}_{acs} H_v(t)$ is returned by *getReferenceTime* interface at any time t such that $\hat{\beta}_{acs} = (\hat{\beta} + \hat{\beta}_{avg})/2$ and $\hat{\alpha}_{acs} = \overline{Y} - \hat{\beta}_{acs}\overline{x}$. Hence, during the calculation of $\hat{\beta}_{acs}$, the recently calculated least-squares slope $\hat{\beta}$ has *more weight* than the other slope values in $\mathcal{SH}$. Since each slope in $\mathcal{SH}$, including $\hat{\beta}$, is an independent estimate of the slope of the real regression line, the variance of $\hat{\beta}_{acs}$ is smaller than that of $\hat{\beta}$ by a factor of $\frac{N+1}{4N}$.

B.1.4 Experimental Results For The Methods Which Use Slope History

For this experiment, we modified FTSP in such a way that in addition to the reference clock estimation by using least-squares, we integrated a slope history with a size of 8 entries and allowed the nodes to estimate the reference clock by using the presented slope methods. As a final modification, we added an interface in order for applications to query the estimated reference time calculated by using these methods. With such modifications to FTSP, we could evaluate all strategies under identical message delays, packet loss rates and environmental conditions.

Figure 5 shows the global/local skew values and the skew values to the reference node, and Figure 6 shows the slope values of the estimated regression line node during the experiments on the line topology of 20 sensor nodes. For comparison of the proposed methods, we considered the skew values just after all of the entries in the slope tables of the nodes are filled, which took approximately 7000 seconds. The summary of these values are presented in Table 4.

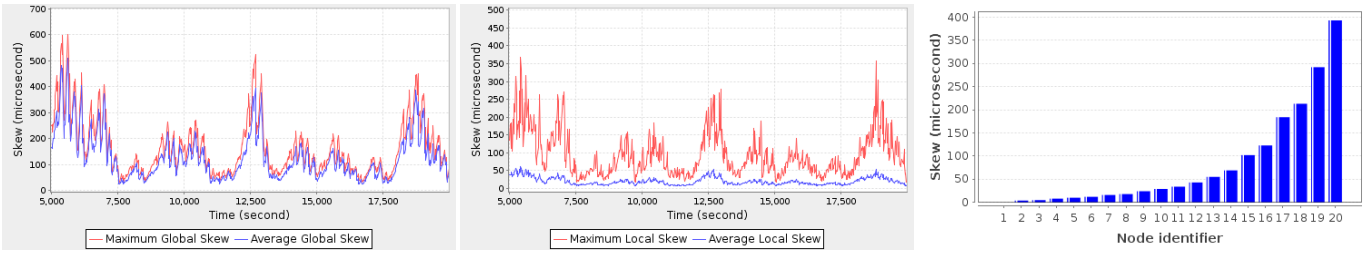
Table 4: Summary of the experimental measurements with FTSP by modifying its least-squares slope estimation.

	LS	**MS**	**AS**	**ACS**
Max. Global Skew	526 μsec	479 μsec	426 μsec	371 μsec
Avg. Global Skew	396 μsec	402 μsec	349 μsec	305 μsec
Max. Local Skew	357 μsec	384 μsec	239 μsec	196 μsec
Avg. Local Skew	54 μsec	52 μsec	39 μsec	34 μsec

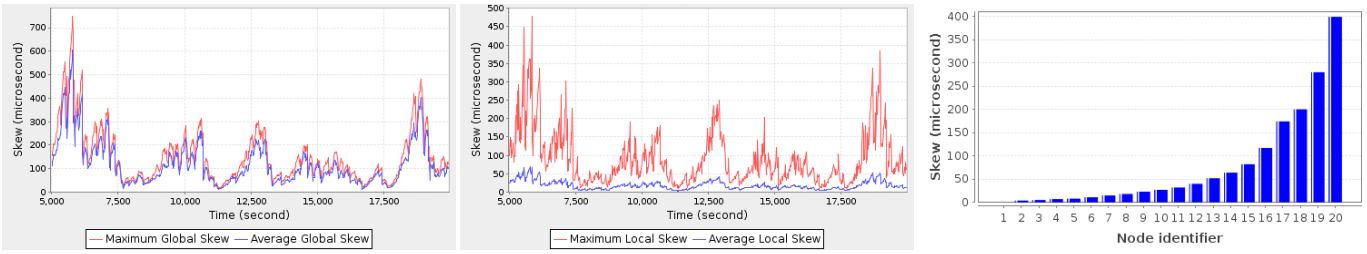
When compared to LS (least squares), MS decreased maximum global and average local skews. On the other hand, we observed increased average global and maximum local skew values. Although the collected slope values were more stable with MS, the maximum skew values to the reference node for the far-away nodes were quite comparable to LS. It can be concluded that the performances of MS and LS are quite similar. We think the reason is that MS decides only one of the values in the slope table as the slope for the estimated regression line and it does not take into account other values in the slope table. Hence, MS suffers from the fact that a more accurate slope value may not be selected since it may not be the median of the slopes in the slope history.

The maximum and average values of the global and local skew as well as the skew between the reference node and far-away nodes were decreased with AS when compared to LS and MS. We observed that at the time instants such that LS exhibited quite variable slope values for the far-away nodes, AS reduced this variation as in MS but the slope values change more smoothly. The superiority of AS over MS and LS is due to the fact that it takes into account all of the slopes in the slope history. On the other hand, we obtained smaller skew and more stable slope values with ACS than with AS. The superiority of ACS over AS is due to the fact that in ACS the recently calculated slope has more weight than the other slopes in the slope history while all of the slopes have equal weights in AS.

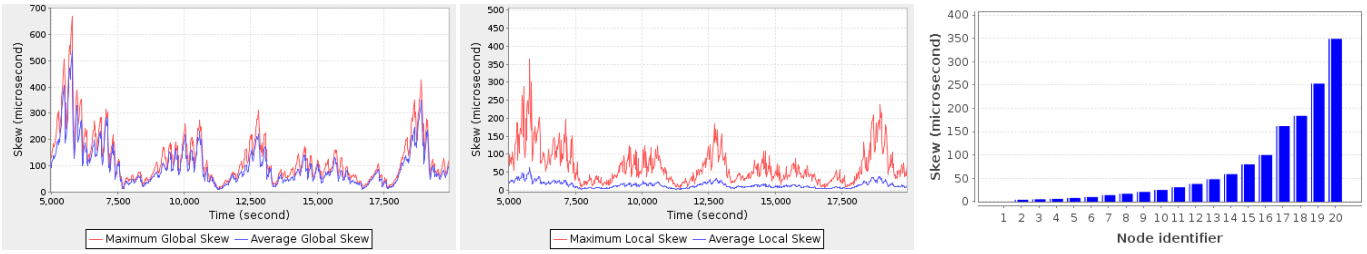
Our experimental results showed that a small modification to FTSP gained us a remarkable synchronization improvement on a line of 20 sensor nodes. Maintaining an extra slope history and applying ACS in order to provide synchronized notion of time to the applications led to increased synchronization quality without changing the propagation speed of the flood. The slope value calculated by ACS is affected by the previously calculated slope values in the slope table and hence data from a larger period are considered. Although this strategy improves the performance of FTSP, it can be observed from Figure 6 that as the distance from the reference node gets larger, the collected slope values show quite variability and the maximum observed skew to the reference node increases exponentially for all proposed methods. Hence, the synchronization performance of these methods are unsatisfactory.



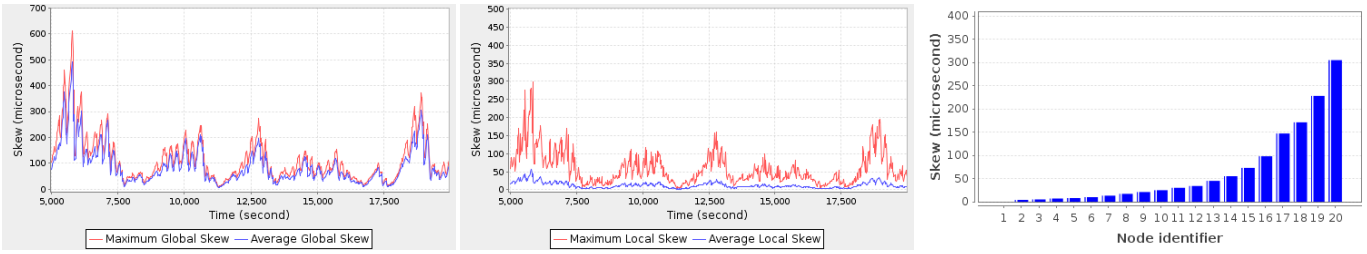
(a) Original FTSP with Least-Squares



(b) FTSP with MS

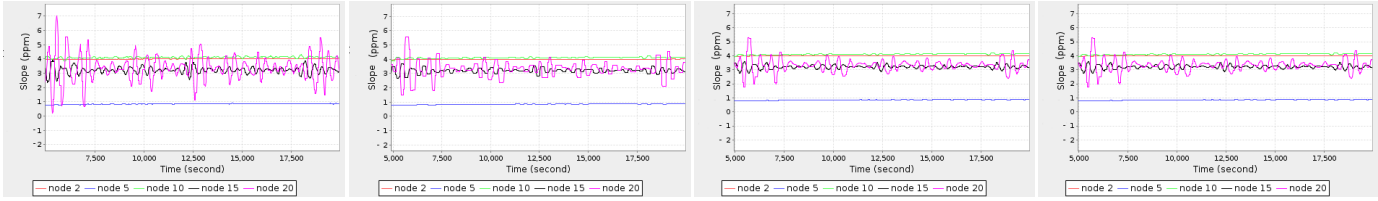


(c) FTSP with AS



(d) FTSP with ACS

Figure 5: Global skew (left column), local skew (middle column) and the maximum synchronization error versus distance from the reference (root) node 1 (right column) on the line topology for FTSP with the proposed methods.



(a) Original FTSP with Least-Squares (b) FTSP with MS (c) FTSP with AS (d) FTSP with ACS

Figure 6: Slope values of the estimated regression line from which 1.0 is subtracted for the nodes running FTSP on the line topology with the proposed methods.

B.2 Slow-Flooding with Rate Dissemination

Although the slope methods are claimed to improve the synchronization accuracy, they are unable to solve the scalability problem of slow-flooding. The synchronization error of the far-away nodes to the reference is still an exponential function of the network diameter since they perform least-squares on the synchronization points whose errors are amplified at each hop due to waiting times. By considering this situation, we realize the following fact.

Fact 4: Least-squares must not be performed on data which are originated by a far-away node in slow-flooding based time synchronization.

In this subsection, we propose Rate Flooding Time Synchronization (RFTS) whose pseudo-code is presented in Algorithm 3. In RFTS, each node not only floods the estimated value but also the estimated relative rate of the reference clock.[7] Moreover, the nodes are restricted to perform least-squares in order to calculate the estimated relative hardware clock rates of their neighbors. Hence, least-squares is performed on data which are originated by a node which is the neighbor of the current node (one hop away).

In order to estimate the clock of the reference node, each node $v \in V$ maintains variables related to its logical clock whose value at any time t is calculated as follows:

$$\begin{aligned} L_v(t) &= base_v(t) \\ &\quad + (H_v(t) - lastUpdate_v(t))\, h_v^r(t). \end{aligned} \quad (28)$$

The variable $base_v$ is required to store the reference clock estimate carried by the most recent synchronization message received and $lastUpdate_v$ holds the hardware clock reading at the time at which $base_v$ variable is updated. The variable h_v^r holds the estimated relative rate of the reference clock. This variable can also be considered as the *rate multiplier* of the logical clock of v. It can be noticed from the equation 28 that L_v is calculated by adding up the amount of progress since the latest update of the base value.

Node v also maintains a repository in order to keep track of the relative hardware clock rates of its neighbors. When v is powered on, the neighbor repository is cleared and the estimated relative hardware clock rates kept for each neighbor are initialized to 1 (Line 2). seq_v variable which stores the largest sequence number received from the reference node, $base_v$ and $lastUpdate_v$ variables are initialized to zero (Line 3). The estimated relative rate of the reference clock h_v^r is set to one (Line 4). Moreover, a periodic timer is started which will fire every time the hardware clock progresses B units (Line 5).

Each time a synchronization message is received from any neighboring node $u \in \mathcal{N}_v$, the (H_v, H_u) pair and the estimated relative hardware clock rate h_v^u is stored in the slot assigned for that neighbor in the repository (Lines 7-8). It should be noted that the slope of the least-squares line which is calculated by using (H_v, H_u) pairs, is an estimate for the relative hardware clock rate h_u/h_v. If the received synchronization message carries a higher sequence number (Line 9), this situation indicates that the reference node has recently flooded its current logical clock value into the network. Hence, v sets the base value of its logical clock to the received logical clock value (Line 10). The hardware clock of v is stored at $lastUpdate_v$ (Line 11) and the sequence number is updated (Line 12). Since v keeps the estimated relative hardware clock rate of u, it can calculate h_v^r by multiplying h_v^u with the received h_u^r value (Line 13).

When a timeout event is generated (Line 16), only the reference node increments its sequence number and initiates a new flooding round (Line 17). Each node broadcasts a message which carries the value of its logical clock, hardware clock, estimated relative rate of the reference clock and sequence number (Line 18).

Algorithm 3 RFTS pseudo-code for node v with a fixed reference node whose node identifier is ROOT.

1: **Initialization**
2: clear repository and $\forall u \in \mathcal{N}_v$ set $h_v^u \leftarrow 1$
3: $base_v \leftarrow 0$; $lastUpdate_v \leftarrow 0$; $seq_v \leftarrow 0$
4: $h_v^r \leftarrow 1$
5: start periodic timer with period B
6:
7: □ **Upon receiving** $< L_u, H_u, h_u^r, seq_u >$
8: store (H_v, H_u) and estimate h_v^u using least-squares
9: **if** $seq_v < seq_u$
10: $base_v \leftarrow L_u$
11: $lastUpdate_v \leftarrow H_v$
12: $seq_v \leftarrow seq_u$
13: $h_v^r \leftarrow h_u^r . h_v^u$
14: **endif**
15:
16: □ **Upon timer timeout**
17: **if** $u = ROOT$ **then** $seq_u \leftarrow seq_u + 1$ **endif**
18: **broadcast** $< L_v, H_v, h_v^r, seq_v >$

B.2.1 Analysis

In RFTS, each node receives messages which carry estimated relative rate of the reference node with respect to the hardware clock rate of the neighboring node from which the message is received. Let v_0, v_1 and v_2 form a line topology. Node v_1 calculates the estimated rate of the reference node v_0 by using independent hardware clock readings from v_0. When v_1 sends its estimate to v_2, v_2 multiplies the received estimate with the estimated hardware clock rate of v_1 which has been also calculated by using independent hardware clock readings from v_1. Hence, the received estimated rate of the reference node from v_1 is independent from the estimated relative hardware clock rate of v_1 with respect to the hardware clock of v_2. In general, the estimated hardware clock

7. The relative rate of the reference clock from node v's perspective is defined as the rate of the reference clock with respect to the hardware clock rate of node v.

rates of the neighboring nodes are also independent from the received estimated relative rate of the reference node. By using this strategy, we show that this drift estimation mechanism also exhibits an exponential error with the network diameter as follows.

Consider the execution of RFTS on a line of n nodes $v_o, v_1, ..., v_{n-1} \in V$ such that v_0 is the reference node. Assume that $\frac{1}{S_{xx}} \leq \frac{1}{\mathcal{S}}$ holds for all times t and for all nodes $v_i \in V$.[8]During the execution, v_1 collects independent hardware clock readings of v_0 which suffer from an error $\varepsilon \sim \mathcal{N}(0, \sigma^2)$. Hence, $h_{v_1}^{v_0} \sim \mathcal{N}(\frac{h_{v_0}}{h_{v_1}}, \kappa_{v_1} = \frac{\sigma^2}{\mathcal{S}})$ holds due to distribution 27. While v_2 collects hardware clock readings of v_1 which suffer from an error $\varepsilon \sim \mathcal{N}(0, \sigma^2)$ in order to estimate $\frac{h_{v_1}}{h_{v_2}}$, it also receives the estimate $h_{v_1}^{v_0}$ from v_1. Due to the independency of the hardware clock readings of v_1, it holds that $h_{v_2}^{v_1} \sim \mathcal{N}(\frac{h_{v_1}}{h_{v_2}}, \frac{\sigma^2}{\mathcal{S}})$. Since v_1 calculates $h_{v_1}^{v_0}$ by considering independent hardware clock values from v_0 and v_2 calculates $h_{v_2}^{v_1}$ by considering independent hardware clock values from v_1, $h_{v_1}^{v_0}$ and $h_{v_2}^{v_1}$ are independent. Hence, we get that:

$$\begin{aligned} Var(h_{v_2}^{v_0}) &= Var(h_{v_1}^{v_0}.h_{v_2}^{v_1}) \\ &= (E[h_{v_1}^{v_0}])^2 Var(h_{v_2}^{v_1}) + (E[h_{v_2}^{v_1}])^2 Var(h_{v_1}^{v_0}) \\ &\quad + Var(h_{v_1}^{v_0}) Var(h_{v_2}^{v_1}) \\ &= (\frac{h_{v_0}}{h_{v_1}})^2 \frac{\sigma^2}{\mathcal{S}} + ((\frac{h_{v_1}}{h_{v_2}})^2 + \frac{\sigma^2}{\mathcal{S}}) \kappa_{v_1} \\ &= \kappa_{v_2}. \end{aligned} \tag{29}$$

$$E[h_{v_2}^{v_0}] = E[h_{v_1}^{v_0}] E[h_{v_2}^{v_1}] = \frac{h_{v_0}}{h_{v_2}}. \tag{30}$$

Thus, we have that:

$$h_{v_2}^{v_0} \sim \mathcal{N}(\frac{h_{v_0}}{h_{v_2}}, \kappa_{v_2}) \tag{31}$$

At each step, $h_{v_k}^{v_0}$ and $h_{v_{k+1}}^{v_k}$ for $1 \leq k \leq n-2$ are independent from each other and by the same construction, the error distribution of $h_{v_{n-1}}^{v_0}$ can be formulated as follows:

$$h_{v_{n-1}}^{v_0} \sim \mathcal{N}(\frac{h_{v_0}}{h_{v_{n-1}}}, \kappa_{v_{n-1}})). \tag{32}$$

Since for all $v_i, v_j \in V$ it holds that $h_{v_i}/h_{v_j} \approx 1$,[9] it follows that $\kappa_{v_i} \approx (\frac{\sigma^2}{\mathcal{S}} + 1)^i - 1$. Thus, we reach the following theorem.

Theorem B.1: In RFTS, the variance of the relative drift estimation of node $v_i \in V$ is upper bounded by $\mathcal{O}\left((\frac{\sigma^2}{\mathcal{S}} + 1)^i - 1\right)$.

8. Since it can be assumed that number of successive message losses in sensor networks can be upper bounded, $1/S_{xx}$ can also be upper bounded due to periodic synchronization and message exchange.

9. Due to the fact that the hardware clocks exhibit a drift between 30 and 100 ppm.

Table 5: Summary of the experimental measurements with RFTS.

Max. Global Skew	68 μsec
Avg. Global Skew	55 μsec
Max. Local Skew	67 μsec
Avg. Local Skew	12 μsec

Fact 5: The effect of waiting times at each node on the synchronization accuracy due to slow-flooding is considerably smaller in RFTS when compared to FTSP.

B.2.2 Experimental Results For RFTS

The global and local skew values, the rate multipliers and also the maximum clock skew observed between the nodes 2 to 20 and the reference node 1 observed during the experiments on the line topology with RFTS are presented in Figure 7. The summary of the experimental measurements is given in Table 5. It can be concluded that the proposed method outperforms FTSP quite significantly although it does not change the propagation speed of the flood, employs an identical message pattern and has the same message complexity. If maximum global skew is considered, RFTS improved the performance of FTSP approximately by a factor of 8. As can be observed, the negative effect of the least-squares regression on the scalability is reduced considerably with rate dissemination in RFTS. Since the rate multipliers are quite stable for RFTS, the estimation errors of the far-away nodes are small when compared to FTSP.

For RFTS, the initial network wide synchronization was achieved at approximately 400th second, which took shorter than FTSP. The global skew of this protocol was 70 μsec at that time, respectively. Since the actual performance of RFTS can be observed as soon as the estimated clock value and clock rate of the reference node have been collected by all of the nodes, fast synchronization is the superiority of RFTS.

APPENDIX C
ANALYSIS OF FCSA

We first introduce the following theorem which states that if communication delay is neglected, all of the sensor nodes agree on a common logical clock speed using the clock speed agreement mechanism.

Theorem C.1: If the graph G remains strongly connected, then $\forall v \in V : \lim_{t \to \infty} (h_v(t).l_v(t)) = speed$.

Proof: The equation 4 can be rewritten as $x(t^+) = M.x(t)$ such that $x(t)$ is an nx1 matrix whose ith entry contains the logical clock speed $h_i(t).l_i(t)$ of the node $i \in V$ at time t and M is a nxn row-stochastic matrix corresponding to the graph G whose a_{ij} entry is defined as $1/|\mathcal{N}_i| + 1$ if $\{i, j\} \in E$ and otherwise zero. The proof is based on the fact that the products of row-stochastic matrices converges if G is strongly connected. The detailed proof can be found in [10], [9]. This result also implies that limiting the number of neighbors of a

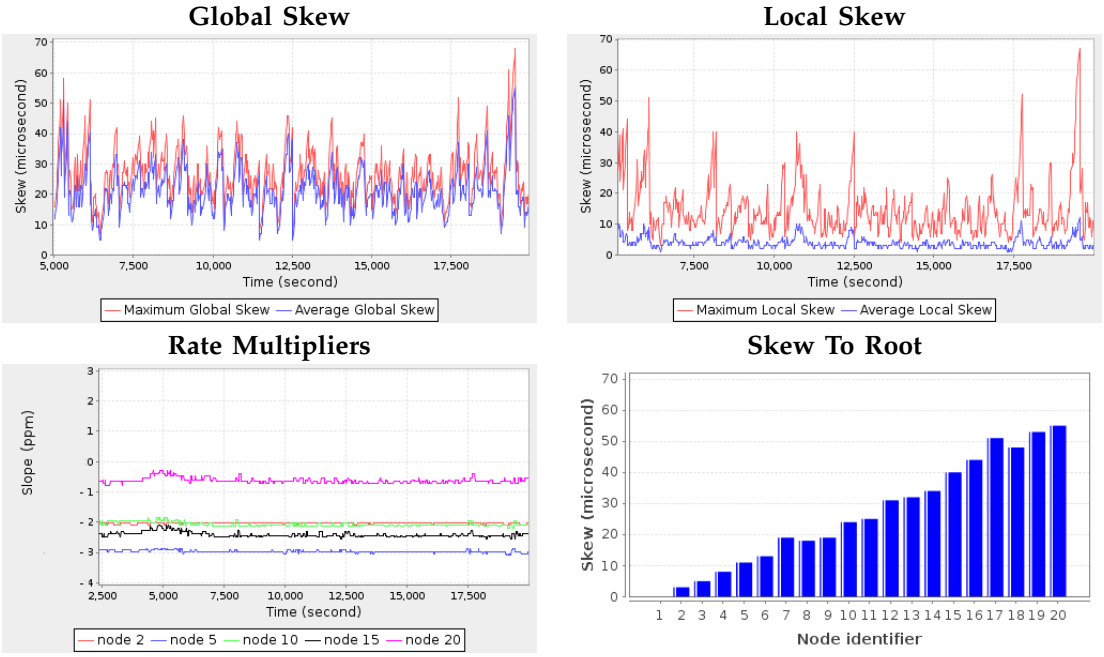


Figure 7: Global skew, local skew, collected rate multipliers from which 1.0 is subtracted and maximum clock skew observed between the nodes 2 to 20 and the reference node 1 measured for nodes running RFTS.

sensor node does not prevent agreement as long as the communication graph remains strongly connected. □

However, since neglecting the uncertainties on the message path is not realistic in WSNs, one must take into account effect of these uncertainties on the agreement. Let the adjacency matrix A and the laplacian matrix L of the network graph $G = (V, E)$ are defined as follows:

$$A(i,j) = \begin{cases} 1, & \{i,j\} \in E \\ 0, & otherwise \end{cases}, \quad (33)$$

$$L = D - A, \quad (34)$$

such that $D = diag(d_1, d_2, ..., d_n)$ is the degree matrix of G, i.e. $d_i = |\mathcal{N}_i|$. Let $0 = \lambda_1(L) \leq \lambda_2(L) \leq ... \leq \lambda_n(L)$ be the eigenvalues of L.

Assume that $\frac{1}{S_{xx}} \leq \frac{1}{\mathcal{S}}$ and $l_{v_i} \leq \ell$ hold for all times t and for all nodes $v_i \in V$. Hence, h_v^u in the equality 4 can be written as

$$h_v^u = \frac{h_u}{h_v} + d_{uv} \sim \mathcal{N}(0, \frac{\sigma^2}{\mathcal{S}}) \quad (35)$$

such that d_{uv} represents the error of the least-squares regression due to the non-deterministic delays occurred during the communication between the nodes u and v. Hence,

$$\begin{aligned} h_v^u.l_u &= \frac{h_u}{h_v}.l_u + d_{uv}.l_u \\ &= \frac{h_u}{h_v}.l_u + \nu_{uv} \end{aligned} \quad (36)$$

holds such that $\nu_{uv} \sim (0, \frac{\sigma^2}{\mathcal{S}}\ell^2)$. By using this equality, the equality 4 can be rewritten as follows:

$$\begin{aligned} l_v(t^+) &= \frac{l_v(t) + \sum_{u\in\mathcal{N}_v} h_v^u(t).l_u(t)}{|\mathcal{N}_v| + 1} \\ &= l_v(t) + \frac{\sum_{u\in\mathcal{N}_v}(\frac{h_u}{h_v}.l_u(t) - l_v(t))}{|\mathcal{N}_v| + 1} \\ &\quad + \frac{\sum_{u\in\mathcal{N}_v}\nu_{uv}(t)}{|\mathcal{N}_v| + 1} \\ &= l_v(t) + c\sum_{u\in\mathcal{N}_v}(\frac{h_u}{h_v}.l_u(t) - l_v(t)) + n_v(t) \end{aligned} \quad (37)$$

where $c = 1/(|\mathcal{N}_v| + 1)$ and $n_v(t) = c\sum_{u\in\mathcal{N}_v}\nu_{uv}(t)$. This equality is the same as the equality 5 in [24][10], hence, the following theorem holds:

Theorem C.2: If the graph G remains strongly connected, then the variance of l_v for any $v \in V$ is upper bounded by

$$\begin{aligned} Var(l_v) &\leq c\frac{\sigma^2}{\mathcal{S}}\ell^2 min\{\mathcal{D}_n max\{\lambda_i\}, \lambda_{max}(A^2)\sum_{i=2}^{n}\lambda_i\} \\ &= \gamma^2, \end{aligned} \quad (38)$$

where $\mathcal{D}_n = \sum_{i=1}^{n} d_i$ is the total degree in the network and $\lambda_i = 1/(2\lambda_i(L) - c\lambda_i^2(L))$.

Proof: Apply Theorem 2 in [24] with $\mathbf{u}$= 0, $\varepsilon = c$ and $\sigma^2 = \frac{\sigma^2}{\mathcal{S}}\ell^2$. □

This result shows that the error of the clock speed agreement essentially depends on the total degree of the network and the eigenvalues of L and A^2. Hence, the

10. Consider equality 4 in [24]. Substitute $t_i(k)$ with $\frac{h_i}{h_i}.l_i(t) = l_i(t)$, $\hat{t}_j(k)$ with $h_i^j.l_j(t) = \frac{h_j}{h_i}.l_j(t) + \nu_{ij}$ and ε with $1/(|\mathcal{N}_v| + 1)$. Hence, we reach equality 5.

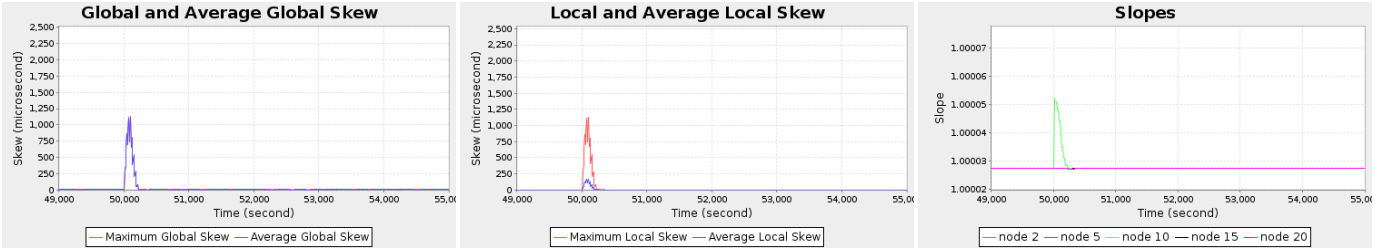
error of the drift estimation mechanism in FCSA does not grow exponentially with the diameter of the network as in FTSP. Let FCSA is executed on the line topology $v_0, v_1, ..., v_n$ such that v_0 is the reference node. Let B be the expected waiting time at each hop to propagate the estimated logical clock value of the reference node. During the execution, v_1 receives the estimated logical clock value of the reference node which is affected by the error introduced by the message delay. Due to the expected waiting time B at node v_1, the variance of this estimate is increased by $\gamma^2 B^2$. Node v_2 receives this estimate with an additional error introduced by the message delay. Similarly, after waiting an expected time of B, the variance of the estimate is increased by $\gamma^2 B^2$. It can be concluded that the variance of the estimated logical clock at node v_n becomes on the order of $\mathcal{O}(\gamma^2 B^2 \mathcal{D})$. Therefore, the standard deviation of the estimated logical clock of the reference node is upper bounded by the square root of the network diameter.

APPENDIX D
REACTION TO DYNAMICS

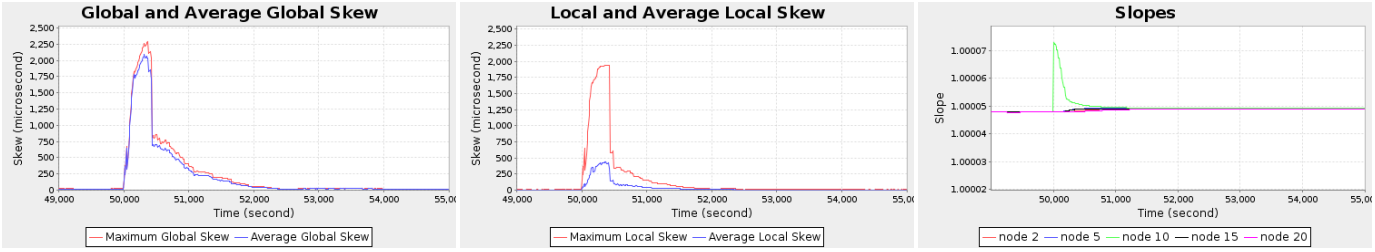
In this section we present simulation results of PulseSync and FCSA in order to observe their adaptation to the changing environmental conditions. Since PulseSync employs fast flooding, it is quite clear that it adapts changing environmental conditions more quickly than FCSA. However, the adaptation time of FCSA is not too long when compared to PulseSync. In order to verify this argument, we performed simulations for PulseSync and FCSA on a line topology of 20 sensor nodes. After the synchronization is established, we suddenly changed the drift rate of the node whose identifier is 10 in order to simulate a change on the environmental temperature. The results of our simulations is presented in Figure 8.

As can be observed, at t=50000 of the simulation, the hardware clock rate of node 10 was changed. After that time, the drift estimation of PulseSync adopted to dynamics in approximately 500 seconds. On the other hand, this adaptation took approximately 1500 seconds for FCSA. If global and local skews are considered, the adaptation time of PulseSync was approximately 500 seconds while it took 2000 seconds for FCSA. After the adaptation, the synchronization accuracies of PulseSync and FCSA were almost the same.

It can be concluded that the delayed adaptation in FCSA occurred due to the slow-flooding, but the adaptation time is not too long when compared to PulseSync. Hence, FCSA can be preferred on also dynamic environmental conditions.



(a) PulseSync



(b) FCSA

Figure 8: Global skew (left column), local skew (middle column) and the slopes/rate multipliers from which 1.0 is subtracted (right column) for PulseSync and FCSA, respectively.