External Gradient Time Synchronization In Wireless Sensor Networks

Kasım Sinan Yıldırım and Aylin Kantarcı

Abstract—Synchronization to an external time source such as Coordinated Universal Time (UTC), i.e. external synchronization, while preserving tight synchronization among neighboring sensor nodes may be crucial for applications such as determining the speed of a moving object in wireless sensor networks. However, existing time synchronization protocols in the literature which can be used for external synchronization poorly synchronize neighboring nodes. On the other hand, the only protocol which aims at optimizing the synchronization error between neighboring nodes is lack of a mechanism which synchronizes sensor nodes to a reference node and hence it cannot provide external synchronization. Therefore, there is a lack in the literature of a time synchronization protocol which can be used by applications demanding both external synchronization and tight synchronization among neighboring nodes.
In this paper, we answer the question of whether it is possible for sensor nodes to synchronize to a reference node while they optimize the clock skew between their neighboring nodes at the same time. Within this context, we present a novel time synchronization protocol, namely External Gradient Time Synchronization Protocol (EGSync). In EGSync, each sensor node synchronizes to a reference node by using time information flooded by this node as well as synchronizes to its neighboring nodes by employing the agreement algorithm. We implemented EGSync on the MICAz platform using TinyOS and evaluated it on a testbed setup including 20 sensor nodes. We present the experimental results on our testbed and the simulation results for networks with larger diameters and densities.

Index Terms—Distributed Algorithms, External Time Synchronization, Gradient Time Synchronization.

1 Introduction

Common notion of time is vital for the correct and efficient operation of sensor nodes forming wireless sensor networks (WSNs). This notion cannot be achieved by using built-in hardware clocks alone since they frequently drift apart. Hence, each sensor node is required to participate in time synchronization and calculate a *software clock* which represents the common time inside the network. The objective of time synchronization is to minimize the differences between the software clocks of the nodes, i.e. *clock skew*, at any time instant [1], [2], [3].

A majority of the time synchronization protocols for WSNs in the literature [1], [4], [5], [6], [7], [8], [9], [10] aim at optimizing the clock skew between arbitrary sensor nodes, i.e. *global skew*, regardless of the distance[1] between them. However, sensor nodes may be poorly synchronized to their neighboring nodes with these protocols since they do not take into account the time information of all of their neighbors. Optimizing the synchronization error between neighboring nodes, i.e. *local skew*, may be crucial for applications such as target tracking in WSNs [11].

Consider a tracking application which requires sensor nodes to report a target object's velocity in units meters/second to a base station. In order to achieve this goal, each sensor node is required to record the time at which it detected the target object and exchange this information with its neighboring nodes. The estimated velocity of the target object can be calculated by dividing the difference of the detection times $\triangle$ by the distance between the sensor nodes. In order to calculate $\triangle$ in terms of real-time passed in units of second, sensor nodes are required to be synchronized with respect to an external time sources such as Coordinated Universal Time (UTC) [12], [13]. As can be observed, the smaller the synchronization error is observed between the neighboring nodes, the more accurate $\triangle$ and hence the velocity is estimated.

Gradient Time Synchronization Protocol (GTSP) [14] is the first and only protocol which aims at optimizing local skew in WSNs. GTSP is a *completely decentralized* protocol since each sensor node synchronizes to its neighboring nodes and there is not any special node which acts as a time reference. Unfortunately, this property leads to inability for GTSP to provide synchronization to an external time source, i.e. *external synchronization* [9]. In WSNs, a reference node which is synchronized to an external time source is required in order to achieve external synchronization. By internally synchronizing the remaining sensor nodes to this reference node, network-wide external synchronization is established. Therefore, there is a lack in the literature of a time synchronization protocol which can be used by applications demanding both external synchronization and local skew optimization, such as the tracking application we mentioned.

The main contribution of this paper is to answer the question of whether it is possible for sensor nodes to synchronize to a reference node while they optimize

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

1. i.e. number of hops

the clock skew between their neighboring nodes at the same time. Within this context, we present a novel time synchronization protocol, namely External Gradient Time Synchronization Protocol (EGSync). In EGSync, each sensor node synchronizes to a reference node by using time information flooded by this node as well as synchronizes to its neighboring nodes by employing an agreement algorithm. Hence, EGSync is able to achieve external synchronization by optimizing the synchronization error among the neighboring nodes.

The remainder of this paper is organized as follows. Section 2 describes the related work on time synchronization in WSNs. In Section 3, we describe our system model. We present and describe EGSync in detail in Section 4. Section 5 presents our implementation of EGSync which has been done in TinyOS[2] and its evaluation on a testbed setup including 20 MICAz sensor nodes. We also present simulation results in this section. Finally, we present our conclusions in Section 6.

2 Related Work

The fundamental problem of time synchronization has been extensively studied for traditional distributed systems. There are several theoretical studies in the literature that focus on bounding the skew between arbitrary nodes, i.e. global skew, in any network [15], [16], [17], [18], [19]. There are also considerable amount of protocol based studies in the literature that focus on optimizing global skew in WSNs [1], [4], [5], [6], [7], [20], [21], [8], [9], [10]. The fundamental shortcoming of these practical studies is that their executions may introduce large synchronization errors between neighboring nodes. As an example, Flooding Time-Synchronization Protocol (FTSP) [7], the de facto time synchronization protocol in WSNs, is designed to optimize global skew. FTSP synchronizes the whole network by electing a reference node based on the smallest node identifier which serves as a time source. The reference node periodically floods its clock through the network. With flooding, an ad-hoc tree is constructed on the fly. However, neighboring nodes which are on different paths of the tree may not be synchronized well because errors propagate down differently on these paths.

Fan and Lynch [11] introduced *gradient clock synchronization* where the clock skew between nodes is a function of their distance and they emphasized the skew between neighboring nodes, i.e. local skew. There are considerable amount of studies that focus on the development and analysis of theoretical algorithms achieving optimal local skew [22], [23], [24], [25], [26], [27], [28], [29]. However, less attention has been paid to the development of practical protocols which aim at optimizing the local skew. Up to our knowledge Gradient Time Synchronization Protocol (GTSP) [14] is the only practical study in WSNs with this aim. In GTSP, synchronization messages received from neighboring nodes are used to adjust clocks. Using a simple algorithm based on averaging the time information received from neighbors (which has previously been introduced in [30], [31]), the sensor nodes agree on a common clock speed and clock value. Although GTSP achieves a better local skew than FTSP, it leaves the question open whether it is possible for sensor nodes to synchronize to a reference time source while they optimize the clock skew between their neighbors at the same time.

3 System Model

In this section, we introduce our system model which we use in the rest of the paper. A WSN can be modeled as a graph $G = (V, E)$ which consists of a set of n sensor nodes represented by vertex set $V = \{1, ..., n\}$ and bidirectional communication links between these nodes represented by $E \subseteq V \times V$, respectively. Any node $u \in V$ can only communicate with its neighboring nodes, which are defined as the nodes to which it is directly connected. $\mathcal{N}_u = \{v \in V | \{u, v\} \in E\}$ and $|\mathcal{N}_u|$ represent the set of neighbors and the number of neighbors of node u, respectively.

The *distance* between any nodes $u, v \in V$ 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. The *adjacency matrix* A and the *laplacian matrix* L of the graph G are defined as follows:

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

$$L = D - A, \quad (2)$$

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

Each node u is assumed to be equipped with an unmodifiable *hardware clock*, which is denoted by $H_u()$. We define the reading of the hardware clock of the node u at real time t as

$$H_u(t) = \int_0^t h_u(\tau) d\tau \quad (3)$$

where $h_u(\tau)$ represents the *rate* (speed) of the hardware clock at time τ. Clock drift occurs where the hardware clock does not progress at the exact speed of real-time. The frequencies of crystal oscillators in WSNs exhibit a drift between 30 and 100 ppm.[3] Therefore, we assume that hardware clocks have *bounded drifts* such that for all times t it holds that

2. http://www.tinyos.net

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

$$1-\varepsilon \leq \quad h_u(t) \quad \leq 1+\varepsilon \tag{4}$$

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*, which is denoted by $L_u()$, in order to acquire synchronized notion of time. The value of the logical clock at time t is calculated as follows:

$$L_u(t) \quad = \quad \int_0^t h_u(\tau) l_u(\tau) d\tau + \theta_u(t) \tag{5}$$

where $l_u(t)$ is called the *rate multiplier* and $\theta_u(t)$ is called the *offset* of the logical clock.

The objective of time synchronization is to minimize the differences of the logical clock values in the system, i.e. *clock skew*. The *global skew* at any time t is defined as the largest clock skew between any two arbitrary nodes, i.e. $\underset{u,v \in V}{max} \{|L_u(t) - L_v(t)|\}$. Similarly, the *local skew* at any time t is defined as the largest clock skew between any two neighboring nodes, i.e. $\underset{u \in V, v \in \mathcal{N}_u}{max} \{|L_u(t) - L_v(t)|\}$.

4 External Gradient Time Synchronization Protocol (EGSync)

In this section, we present External Gradient Time Synchronization Protocol (EGSync) which synchronizes sensor nodes to the clock of a reference node while optimizing synchronization error between neighboring nodes at the same time. In order to achieve this goal, EGSync executes an agreement protocol among the neighboring sensor nodes and forces them to agree on a common clock value and clock speed. Additionally a reference node floods its time information into the network. By using the flooded information together with the common clock value and clock speed, the other nodes agree on the clock speed and value of the reference node. Figure 1 presents the general strategy of EGSync on a sample network.

The pseudo-code of the EGSync protocol is presented in Algorithm 1. Sensor nodes executing this algorithm agree on the hardware clock speed and hardware clock value of a predefined reference node ref, whose node identifier equals to $ROOT$. Each sensor node $v \in V$ maintains a neighbor repository in order to keep track of time information of its neighbors. By using this repository, node v can estimate at any time the relative hardware clock rate (with respect to its hardware clock) and the logical clock value of each of its neighboring nodes. 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 which a sensor node keeps track of is limited. In order to keep track of the

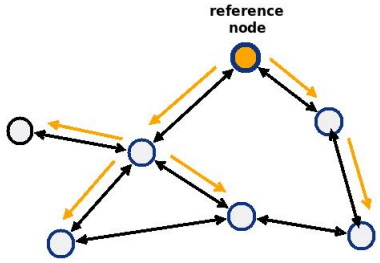

Figure 1: The general strategy of EGSync: Nodes agree on a common clock value and speed by executing an agreement protocol with their neighbors. Additionally, a reference node floods its rate multiplier and time offset in order for the other nodes to agree on its clock speed and value.

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

1: **Initialization**
2: $\forall u \in \mathcal{N}_v$ set $h_v^u \leftarrow 1$, $l_v^u \leftarrow 1$, $L_v^u \leftarrow 0$
3: $l_v \leftarrow 1$, $\theta_v \leftarrow 0$
4: $l_v^{ref} \leftarrow 1$, $\Delta_v^{ref} \leftarrow 0$, $seq_v \leftarrow 0$
5: set periodic timer with period B
6:
7: □ **Upon receiving** $< H_u, L_u, l_u, l_u^{ref}, \Delta_u^{ref}, seq_u >$
8: store (H_v, H_u) pair and estimate h_v^u
9: $l_v^u \leftarrow l_u$
10: $L_v^u \leftarrow L_u$
11: update l_v and θ_v
12: **if** $seq_v < seq_u$
13: $l_v^{ref} \leftarrow l_u^{ref}$
14: $\Delta_v^{ref} \leftarrow \Delta_u^{ref}$
15: $seq_v \leftarrow seq_u$
16: **endif**
17
18: □ **Upon timer timeout**
19: **if** $(node\, identifier = ROOT)$
20: $l_v^{ref} \leftarrow l_v$
21: $\Delta_v^{ref} \leftarrow H_v - L_v$
22: $seq_v \leftarrow seq_v + 1$
23: **endif**
24: **broadcast** $< H_v, L_v, l_v, l_v^{ref}, \Delta_v^{ref}, seq_v >$

latest time information received from the reference node ref, node v maintains three extra variables: l_v^{ref} holds the latest rate multiplier, Δ_v^{ref} holds the latest difference between the hardware clock and the logical clock, i.e. *time offset* and seq_v holds the largest sequence number received from the reference node, respectively.

When node v is powered on, the time information kept for each neighboring node $u \in \mathcal{N}_v$ are initialized: the relative hardware clock rate h_v^u is initialized to 1, the rate multiplier l_v^u is initialized to 1 and the estimated logical clock value L_v^u is initialized to 0 (Line 2). Additionally, node v initializes its rate multiplier l_v to 1 and its logical clock offset θ_v to zero (Line 3). Hence, the value of the logical clock of node v at any time is equal to its hardware clock reading until the values of the parameters l_v and θ_v are changed upon

receiving synchronization messages from neighboring nodes. Moreover, l_v^{ref} is initialized to 1 and Δ_v^{ref} and seq_v are initialized to zero (Line 4). Last, node v starts a periodic timer which will fire each time its hardware clock progresses B units (Line 5).

Each time a synchronization message of the form $< H_u, L_u, l_u, l_u^{ref}, \Delta_u^{ref}, seq_u >$ is received from any neighboring node $u \in \mathcal{N}_v$ (Line 7), node v stores the received hardware clock reading H_u and its hardware clock reading H_v at the receipt time of this message as a pair (H_v, H_u) in the regression table assigned for that neighbor. This table has a limited capacity to hold the most recent N pairs. Node v assumes a *linear relationship* between its hardware clock and the hardware clock of node u and performs least-squares regression on the received pairs in order to calculate the *estimated regression line* (Line 8). It should be noted that the slope of this line is an estimate for the relative hardware clock rate h_u/h_v, which we denote by h_v^u. The received rate multiplier and logical clock value are also stored for that neighbor (Lines 9-10). Using these information, node v is able to calculate the estimated logical clock value of node u at any time t, i.e. $L_v^u(t)$, by progressing the received logical clock value L_u at the speed $h_v^u.l_v^u$ with respect to its hardware clock.

Using the values of h_v^u and l_v^u stored for each neighbor $u \in \mathcal{N}_v$, node v updates its rate multiplier and logical clock offset in equality 5 as follows (Line 11):

$$l_v(t^+) = \frac{l_v(t) + \sum_{u \in \mathcal{N}_v} h_v^u(t).l_v^u(t)}{|\mathcal{N}_v| + 1}, \quad (6)$$

$$\theta_v(t^+) = \theta_v(t) + \frac{\sum_{u \in \mathcal{N}_v} (L_v^u(t) - L_v(t))}{|\mathcal{N}_v| + 1} \quad (7)$$

such that t^+ denotes the time just after the update operation. It can be proven that sensor nodes agree on a common logical clock speed and clock value by applying these equations.[4]

If the received synchronization message carries a higher sequence number (Line 12), this situation indicates that the reference node has recently flooded its current rate multiplier and time offset into the network. Hence, v updates l_v^{ref}, Δ_v^{ref} and seq_v to the corresponding received values (Lines 13-16).

For all times t after the clock speed and clock value agreement is achieved, it theoretically holds for the reference node ref and for the node v that $h_{ref}(t).l_{ref}(t) = h_v(t).l_v(t)$. Thus, it follows that

$$h_{ref}(t) = h_v(t).\frac{l_v(t)}{l_{ref}(t)}. \quad (8)$$

Hence, in order to progress its logical clock at the hardware clock rate of the reference node, node v calculates its logical clock using the following equation instead of equation 5:

4. Please see Section B of the Appendix.

Algorithm 2 *getReferenceClock* interface for node v in EGSync, which returns the estimated clock value of the reference node.

1: getReferenceClock()
2: **return** $L_v + \Delta_v^{ref}$

$$L_v(t) = \int_0^t h_v(\tau)\frac{l_v(\tau)}{l_v^{ref}(\tau)}d\tau + \theta_v(t). \quad (9)$$

When a timeout event is generated by the system (Line 18), l_v^{ref} is set to the rate multiplier, Δ_v^{ref} is recalculated by subtracting the value of the logical clock from the value of the hardware clock and the sequence number is incremented if that node is the reference node (Lines 19-23). Otherwise, these parameters remain unchanged. It should be noted that with the increment of the sequence number, a new flooding round is initiated by the reference node. Each node broadcasts a message which carries the value of its hardware clock, logical clock, rate multiplier, rate multiplier and time offset of the reference node and the sequence number (Line 24).

By using the synchronized logical clock values, node v is also able to output the hardware clock value of the reference node, which we denote by H_v^{ref}, as follows:

$$H_v^{ref}(t) = L_v(t) + \Delta_v^{ref}(t). \quad (10)$$

Hence, the time offset received from the reference node is added to the logical clock value in order to calculate this value, as presented by *getReferenceTime()* interface of Algorithm 2 which returns H_v^{ref}.[5]

Due to the non-deterministic delays occurred during the exchange of time information [7], [1], [5], [14], [32], the agreement mechanism employed by EGSync is unable to provide perfect agreement. It can be shown that the error of the agreement is bounded and it essentially depends on several values related to network parameters. Consequently, $L_{ref}(t) - L_v(t)$ is bounded for all time instants t after the agreement is established.

Theorem 4.1: If the graph G representing the communication network remains strongly connected, then $\forall v \in V : \lim_{t \to \infty} (L_{ref}(t) - L_v(t)) \leq \gamma$ holds such that γ depends on the total degree of G and the eigenvalues of L and A^2.

Proof: Please see section B of the Appendix for the details. □

5 Testbed Experiments and Simulations

In this section, we evaluate the performance of EGSync through the experiments performed on a real hardware platform and the simulations performed in

5. EGSync is able to synchronize sensor nodes in the network to an external time source. Please see Section C of the Appendix for the details.

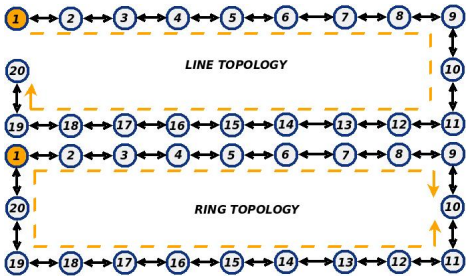

Figure 2: The placement of sensor nodes in order to construct the line and ring topologies for the experiments.

our WSN simulator. For performance comparison, we considered GTSP since its objective is to provide tight synchronization among the neighboring nodes, as in EGSync. We also considered FTSP since it requires a reference node which acts as a time source for the other nodes in the network, it has a publicly available implementation in TinyOS and it is used as a benchmark by most of the studies in the literature [14], [8], [9], [10].

For the evaluation of these protocols, we focused on the instantaneous *global skew* and *local skew* between sensor nodes. We also considered *average global skew* which is defined as the instantaneous average of the global skew and *average local skew* which is defined as the instantaneous average of the local skew by considering all nodes. Moreover, we made a comparison of these protocols in terms of their computation and memory requirements and message complexities.

Table 1: Summary of the measured skew values observed with FTSP, GTSP and EGSync during the experiments.

	Line Topology			Ring Topology		
	FTSP	GTSP	EGSync	FTSP	GTSP	EGSync
Max. Global	518 μs	34 μs	35 μs	38 μs	20 μs	19 μs
Avg. Global	422 μs	26 μs	29 μs	30 μs	15 μs	14 μs
Max. Local	437 μs	10 μs	14 μs	26 μs	10 μs	10 μs
Avg. Local	55 μs	4 μs	5 μs	6 μs	3 μs	4 μs

5.1 Experimental Results

We used a testbed of 20 MICAz sensor nodes in our experiments.[6] We compiled the same application code with FTSP, GTSP and EGSync protocols, and executed these applications on the identical testbed. The beacon period of these protocols was 30 seconds and the number of entries in each regression table was 8. The duration of the experiments was approximately 20000 seconds and we powered on sensor nodes randomly in the first 3 minutes. The placement of sensor nodes during the experiments is shown in Figure 2. The experiments on the ring topology allowed us to observe the synchronization error between neighboring nodes on the different paths of the ad-hoc tree constructed by flooding. The experiments on the line topology, with which a larger diameter is obtained when compared to the ring topology, allowed us to observe the synchronization error between neighboring nodes as the diameter of the network increases.

6. Please see Section A of the Appendix for details on MICAz hardware platform, implementation of the protocols and testbed setup.

5.1.1 Synchronization Performance

We did not take into account the skew values during approximately the first 2500 seconds of the experiments until the initial synchronization process completes. Table 1 summarizes the maximum skew values observed during our experiments.

Figures 3, 4, 5 and 6 show global, average global, local and average local skews measured for FTSP, GTSP and EGSync on the line and on the ring topologies, respectively. On the line topology, EGSync outperformed FTSP dramatically in terms of local and global skew. Moreover, the performances of EGSync and GTSP were quite similar. The same situation can also be observed on the ring topology. When compared to FTSP and GTSP, it can be concluded that EGSync eliminated the deficiency of inability to provide synchronization to a reference node with tight local skew as well as tight global skew.

Figures 7 and 8 present the maximum synchronization error between the reference node and the other nodes on the line and ring topologies for FTSP and EGSync. As can be observed, there is an exponential growth of the synchronization error between the reference node and other nodes as the distance to the reference node gets larger on the line topology. This fact has been previously introduced in [8], which is consistent with our experimental results with FTSP. However, with EGSync, since all nodes agree on the clock value and the clock speed of the reference node, the synchronization errors of far-away and nearby nodes to the reference are quite similar. Hence, we conclude that the serious scalability problem for FTSP does not hold for EGSync. As shown in Figure 2, two subtrees rooted at node 1 is formed due to flooding and hence nodes 8 to 12 have maximum distance to the reference node when compared to the other nodes on the ring topology. In FTSP, these nodes exhibit the largest synchronization error to the reference node. However, in EGSync, the synchronization error between the reference node and the other nodes are quite stable, as those on the line topology.

The main objective of EGSync is to provide tight synchronization between neighboring nodes while synchronizing them to the clock of the reference node. Figures 9 and 10 present the synchronization error between the nodes 19 and 20 which are at the end of the line topology and the maximum local skew per node, respectively. With FTSP, as the distance from the reference node increases, the skew between neighboring nodes also grows. We observed a maximum local skew more than 400 μsec for node with identifier 20 during the experiments. However, the maximum local skews of sensor nodes observed with GTSP and EGSync are quite close to each other and they do not increase drastically

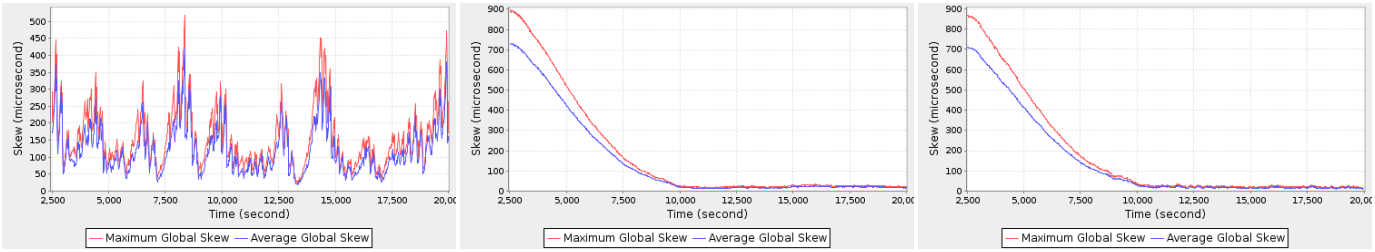

Figure 3: Global and average global skews measured for FTSP (left), GTSP (middle) and EGSync (right) on the line topology, respectively.

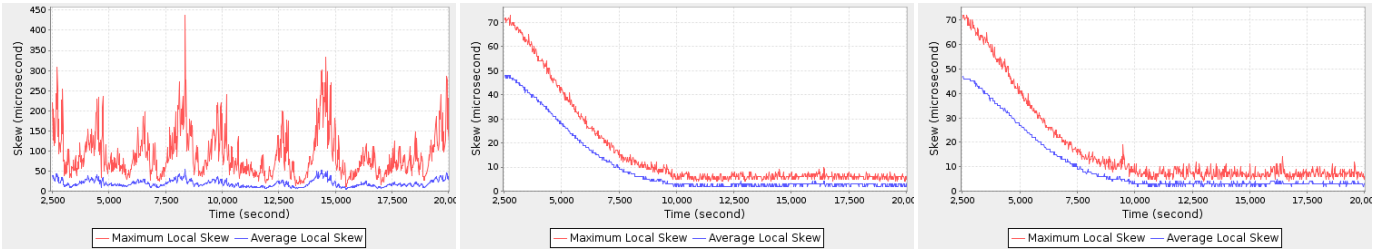

Figure 4: Local and average local skews measured for FTSP (left), GTSP (middle) and EGSync (right) on the line topology, respectively.

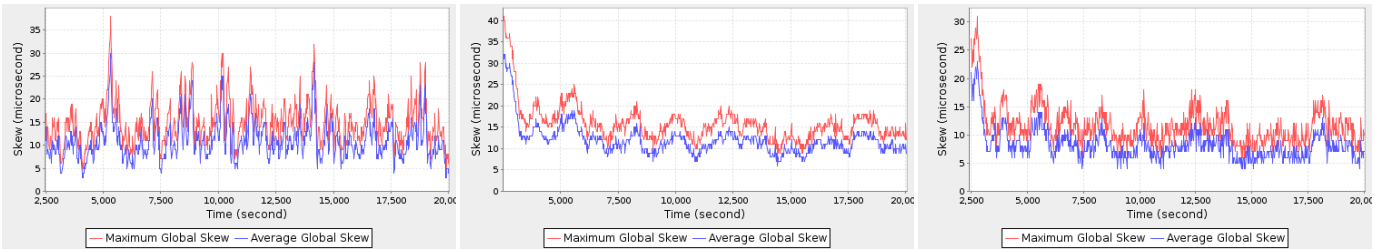

Figure 5: Global and average global skews measured for FTSP (left), GTSP (middle) and EGSync (right) on the ring topology, respectively.

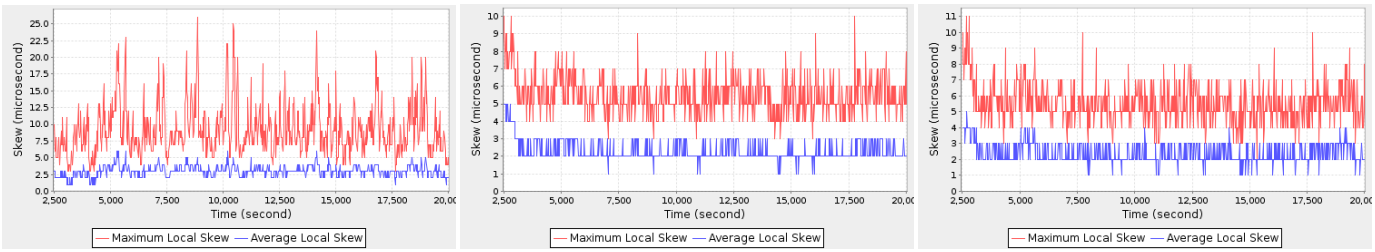

Figure 6: Local and average local skews measured for FTSP (left), GTSP (middle) and EGSync (right) on the ring topology, respectively.

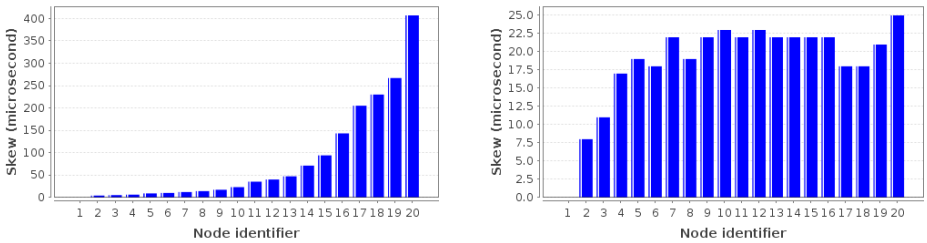

Figure 7: Maximum skew to the reference node (node with identifier 1) per node measured for FTSP (left column) and EGSync (right column) on the line topology.

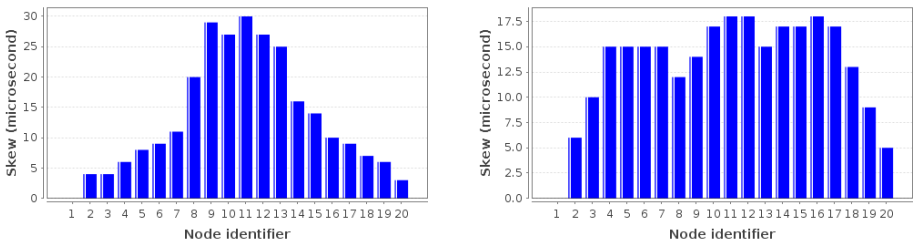

Figure 8: Maximum skew to the reference node (node with identifier 1) per node measured for FTSP (left column) and EGSync (right column) on the ring topology.

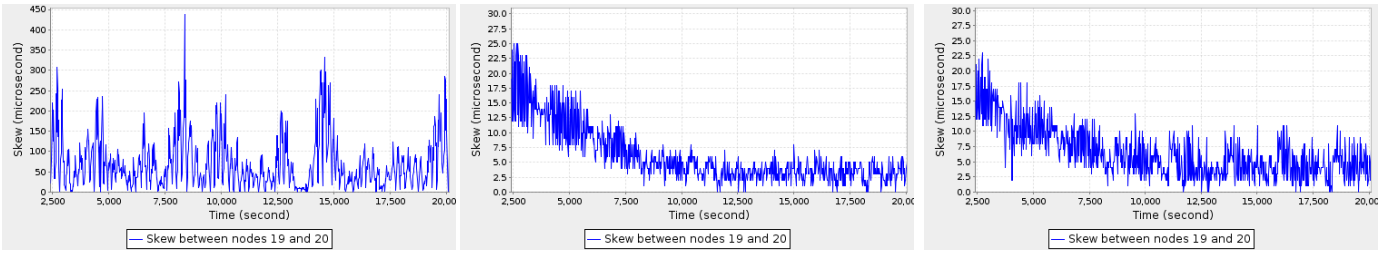

Figure 9: The synchronization error between nodes 19 and 20 for FTSP (left), GTSP (middle) and EGSync (right) on the line topology, respectively.

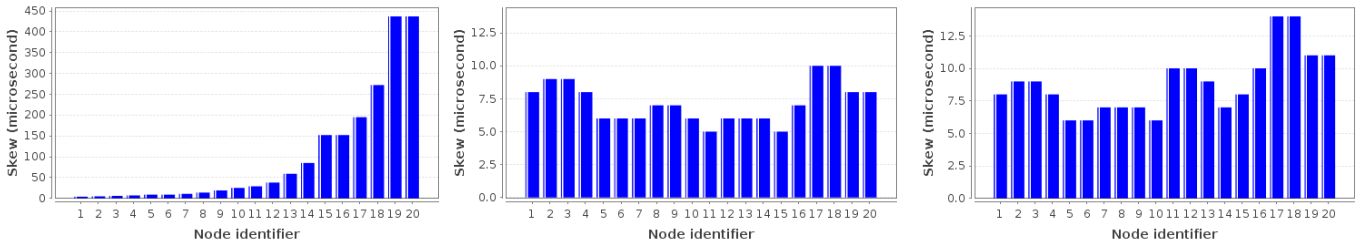

Figure 10: The maximum local skew per node for FTSP (left), GTSP (middle) and EGSync (right) on the line topology, respectively.

as the distance to the reference node increases. This situation can also be observed from the synchronization error between the nodes 19 and 20, which are at the end of the line topology. In FTSP, since nodes consider only the clock values flooded by the reference node and they do not consider the clock of their neighbors, the synchronization error between the nodes 19 and 20 can be quite large during the execution. However, the maximum synchronization error between nodes 19 and 20 is quite small in GTSP and EGSync when compared to FTSP. This is due to the fact that sensor nodes executing GTSP and EGSync synchronize to their neighbors. Although EGSync additionally synchronizes sensor nodes to a reference time source in contrary to GTSP, the synchronization error between the neighboring nodes is quite comparable to that in GTSP.

Figures 11 and 12 present the synchronization error between nodes 9 and 10 on the ring topology and the maximum local skew per node, respectively. It can be observed from this figure that the maximum local skew increases in FTSP as the distance from the reference node increases. However, this case does not hold for GTSP and EGSync. If we consider nodes 9 and 10, although they are neighbors, a local skew of 25 μsec is observed between them in FTSP. However, a skew of 5 μsec and 8 μsec are observed between them in GTSP and EGSync, respectively. The large skew in FTSP is due to the fact that nodes 9 and 10 are on different paths of the tree formed by flooding and the synchronization errors propagate differently on these paths. We conclude that neighboring nodes are more tightly synchronized in EGSync when compared to FTSP and the maximum local skews of the nodes are quite close to each other.

Our experiments showed that EGSync synchronizes sensor nodes to a reference node quite tightly by also preserving tight synchronization among the neighboring nodes.

5.1.2 *Evaluation of the Flooding Mechanism*

As can be observed from the experiments, the local and global skew values of EGSync are very slightly worse than those of GTSP. The reason behind this observation can be explained by examining the periodic time offset flooding mechanism initiated by the reference node. Upon receiving the flooded time offset, nodes update their time information about the reference node. However, nodes propagate the received time offset after waiting for a given period of time. This *slow-flooding* strategy introduces larger instantaneous local and global skews between the nodes which collected the recent time information of the reference node and the nodes which have not collected this time information yet.

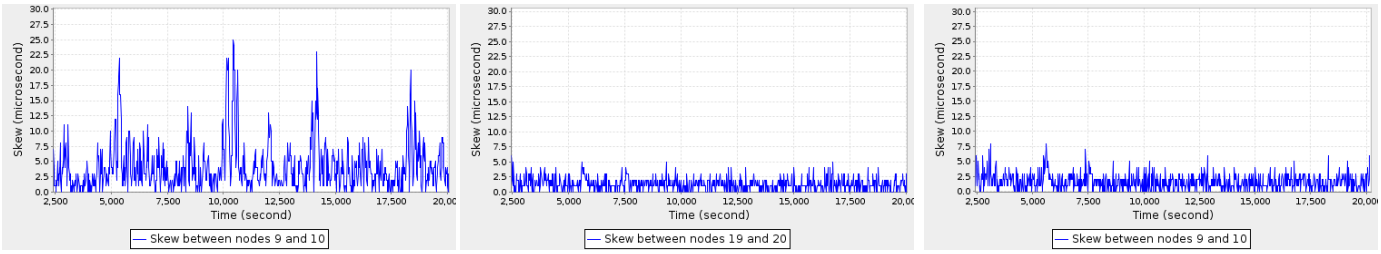

Figure 11: The synchronization error between nodes 9 and 10 for FTSP (left), GTSP (middle) and EGSync on the ring topology, (right) respectively.

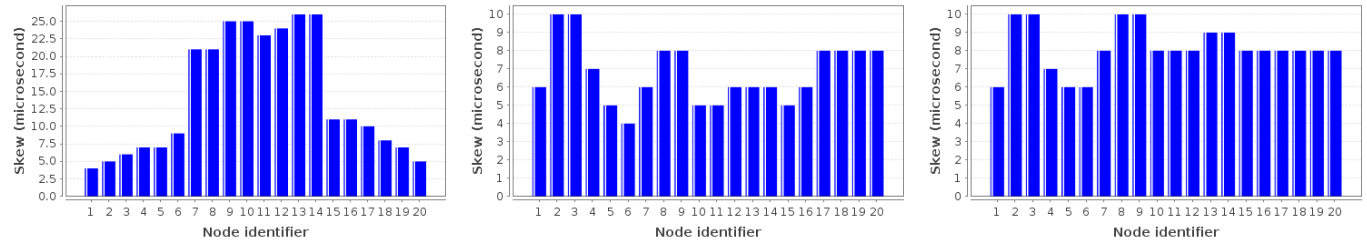

Figure 12: The maximum local skew per node for FTSP (left), GTSP (middle) and EGSync (right) on the ring topology, respectively.

The negative effect of the slow propagation of the reference time information can be reduced by increasing the speed of the flood, i.e. with *rapid-flooding*. However, rapid-flooding can also be slow due to neighborhood contention since the nodes cannot propagate the flood until their neighbors have finished their transmissions and it is difficult in WSNs [33], [10].

5.1.3 Energy Consumption and Memory Requirements

Apart from the synchronization quality, we also compared the performances of EGSync, GTSP and FTSP in terms of memory allocation and communication and CPU overhead. Table 2 summarizes our measurements. The amount of memory which is allocated to store collected time information determines the major memory requirements of the protocols. In publicly available implementation of FTSP, 40 bytes of memory is allocated for the least-squares table which is used to store the time information of the reference node. In our implementation of GTSP, a least-squares table is allocated for each neighbor and hence together with the additional stored information, a total of 64 bytes of memory is allocated to keep track of a neighboring node. EGSync requires an additional 8 bytes for each neighbor (to store information of a neighboring node about the reference node) and hence allocates 72 bytes per neighbor. It can be concluded that GTSP and EGSync increases the memory requirements of time synchronization.

Normally, the longer the synchronization messages, the more the time is required to transmit and receive these messages which increases the energy consumption. The length of the synchronization messages is 13 bytes and 14 bytes in GTSP and FTSP, respectively. Since the time information of the reference node is also propagated in EGSync, this increased the length of the synchronization messages to 23 bytes. However, the payload of the messages is fixed in most platforms, e.g. it is 28 bytes in TinyOS. Hence, synchronization messages of EGSync do not exceed the fixed message length in TinyOS and do not increase the duration of the communication. Moreover, the communication frequency in all of these three protocols is fixed (the beacon period was 30 seconds in our experiments), i.e. their communication complexities are equal. Consequently, it can be concluded that their overhead in terms of communication are equal.

From the point of time where a synchronization message is received to the time it is processed and time information is updated determines the major computation requirements of the protocol and it is another important factor in terms of energy consumption. Upon receiving a synchronization message, FTSP, GTSP and EGSync consumed at most 5500, 6800 and 6950 microseconds of CPU time, respectively, for processing the corresponding message. Hence, EGSync has approximately 23% and 2% more CPU overhead when compared to FTSP and GTSP, respectively.

By considering the overall results, it can be concluded that EGSync slightly increases the energy consumption and memory allocation of GTSP.

Table 2: Memory requirements, CPU overhead and synchronization message length of FTSP, GTSP and EGSync during the experiments. $|\mathcal{N}|$ is used to denote maximum neighborhood cardinality.

	FTSP	GTSP	EGSync
Memory Requirements	40 $bytes$	64*$\|\mathcal{N}\|$ $bytes$	72*$\|\mathcal{N}\|$ $bytes$
CPU overhead	5500 μs	6800 μs	6950 μs
Message Length	13 $bytes$	14 $bytes$	23 $bytes$

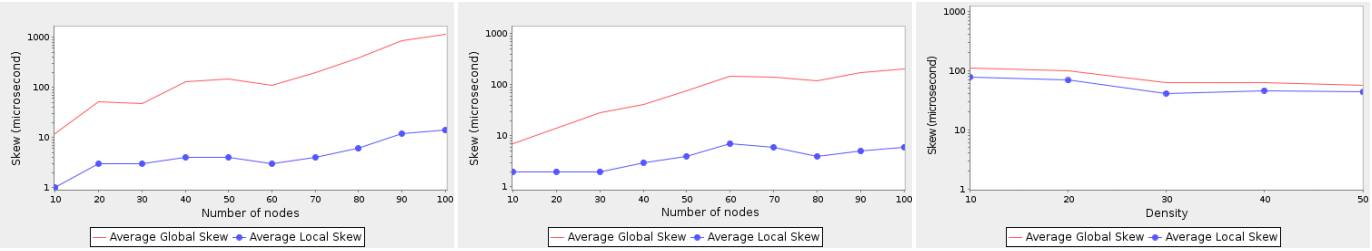

Figure 13: EGSync simulation results for different size of networks with line (left) and ring (middle) topologies. On the right, simulation results of networks consisting 100 nodes which have different neighbor densities are presented.

5.2 Simulation Results for Longer and Denser Networks

In addition to the real-world experiments, we implemented EGSync in our WSN simulator using Java language in order to observe how synchronization errors between arbitrary nodes and between neighboring nodes change as the diameter of the network increases. During our simulations, we modeled the hardware clocks of nodes in software with a random drift of 50 ppm. We applied our evaluation metrics for the networks which is constructed as line and ring topologies which have different number of sensors (and hence diameters). For each network, we performed 10 simulation runs and averaged the calculated average global and average local skews for these runs. Figure 13 shows the simulation results from which we conclude that the local skew and global skew of EGSync grows substantially slowly as the diameter of the network increases. EGSync exhibits quite tight synchronization on longer and larger networks as well.

In order to observe the synchronization accuracy of EGSync on denser networks, we constructed five different networks consisting of 100 sensor nodes and having an average neighborhood cardinality of 10, 20, 30, 40 and 50, respectively. These networks may introduce more packet collisions due to their high density when compared to the line and ring topologies. On the other hand, sensor nodes interact with more neighbors and we observed that this situation improves the performance of the agreement protocol. Moreover, since there are much more alternative paths and the synchronization messages from the reference node may follow shorter paths to reach far-away nodes, these nodes may collect time information more quickly. As can be observed from Figure 13, the synchronization performance of EGSync is affected positively from the increase of the network density.

6 Conclusion

Most of the time synchronization protocols in WSNs suffer from exhibiting large synchronization errors between neighboring nodes although they provide tight synchronization between arbitrary nodes. For instance in Flooding Time-Synchronization Protocol (FTSP), neighboring nodes are poorly synchronized since each sensor node neglects the time information of its neighboring nodes while synchronizing to a reference node. Gradient Time Synchronization Protocol (GTSP) prevents this shortcoming by synchronizing each sensor node to its neighboring nodes. However, it is quite open how to anchor the sensor network to an external time source since there is not any reference node in GTSP. External synchronization is crucial for the correct operation of the applications such as target tracking which also require tightly synchronized neighboring nodes.

In this study, we addressed this problem and designed a time synchronization protocol, namely External Gradient Time Synchronization Protocol (EGSync), which aims at providing tight synchronization between neighboring nodes while synchronizing them to the clock of a reference node at the same time. In EGSync, all of the sensor nodes agree on the clock speed and the clock value of the reference node by using the time information flooded by this node and by applying an agreement algorithm which is based on averaging the time information received from neighboring nodes. Our experiments and simulations showed that the synchronization accuracy of EGSync is quite tight in terms of local and global skew. Moreover, we observed that EGSync has similar overhead in terms of computation, communication and memory requirements when compared to GTSP.

A major disadvantage of EGSync is that, our current implementation requires a fixed reference node and it is predefined before deployment of the sensor network. Hence, EGSync fails to maintain synchronization to the reference node when this node crashes. However, EGSync still continues to synchronize the logical clocks of the sensor nodes by executing the agreement algorithm and by using the time information flooded by the reference node just before the time it has crashed. When using EGSync in order to synchronize the network to UTC time, more than one reference nodes which have access to UTC time can be used to improve the robustness of EGSync in case of reference node failures.

7 Acknowledgments

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] 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.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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. [Online]. Available: http://doi.acm.org/10.1145/1869983.1870010

[11] R. Fan and N. Lynch, "Gradient clock synchronization," *Distrib. Comput.*, vol. 18, no. 4, pp. 255–266, 2006.

[12] J. E. Elson, "Time synchronization in wireless sensor networks," Ph.D. dissertation, 2003, aAI3104330.

[13] D. Tulone, "On the feasibility of time estimation under isolation conditions in wireless sensor networks," *Algorithmica*, vol. 49, pp. 386–411, 2007, 10.1007/s00453-007-9099-1. [Online]. Available: http://dx.doi.org/10.1007/s00453-007-9099-1

[14] 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.

[15] J. Lundelius and N. A. Lynch, "An upper and lower bound for clock synchronization," *Information and Control*, vol. 62, no. 2/3, pp. 190–204, 1984.

[16] J. Y. Halpern, N. Megiddo, and A. A. Munshi, "Optimal precision in the presence of uncertainty," in *STOC '85: Proceedings of the seventeenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1985, pp. 346–355.

[17] S. Biaz and J. L. Welch, "Closed form bounds for clock synchronization under simple uncertainty assumptions," *Elsevier Information Processing Letters*, vol. 80, pp. 151–157, 2001.

[18] B. Patt-Shamir and S. Rajsbaum, "A theory of clock synchronization (extended abstract)," in *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1994, pp. 810–819.

[19] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J. ACM*, vol. 34, no. 3, pp. 626–645, 1987.

[20] 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.

[21] 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.

[22] R. Fan, I. Chakraborty, and N. A. Lynch, "Clock synchronization for wireless networks," *Lecture Notes in Computer Science*, vol. 3544, pp. 400–414, 2005.

[23] T. Locher and R. Wattenhofer, "Oblivious Gradient Clock Synchronization," in *20th International Symposium on Distributed Computing (DISC), Stockholm, Sweden*, September 2006.

[24] C. Lenzen, T. Locher, and R. Wattenhofer, "Clock Synchronization with Bounded Global and Local Skew," in *49th Annual IEEE Symposium on Foundations of Computer Science (FOCS), Philadelphia, Pennsylvania, USA*, October 2008.

[25] R. M. Pussente and V. C. Barbosa, "An algorithm for clock synchronization with the gradient property in sensor networks," *J. Parallel Distrib. Comput.*, vol. 69, no. 3, pp. 261–265, 2009.

[26] F. Kuhn and R. Oshman, "Gradient clock synchronization using reference broadcasts," *CoRR*, vol. abs/0905.3454, 2009.

[27] C. Lenzen, T. Locher, and R. Wattenhofer, "Tight Bounds for Clock Synchronization," in *Journal of the ACM, Volume 57, Number 2*, January 2010.

[28] F. Kuhn, T. Locher, and R. Oshman, "Gradient Clock Synchronization in Dynamic Networks," in *21st ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Calgary, Canada*, August 2009.

[29] F. Kuhn, C. Lenzen, T. Locher, and R. Oshman, "Optimal Gradient Clock Synchronization in Dynamic Networks," in *29th Symposium on Principles of Distributed Computing (PODC), Zurich, Switzerland*, July 2010.

[30] L. Schenato and G. Gamba, "A distributed consensus protocol for clock synchronization in wireless sensor network," in *IEEE Conference on Decision and Control (CDC 07)*, 2007.

[31] 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.

[32] K. Römer, P. Blum, and L. Meier, "Time synchronization and calibration in wireless sensor networks," in *Handbook of Sensor Networks: Algorithms and Architectures*, I. Stojmenovic, Ed. John Wiley & Sons, 2005, pp. 199–237.

[33] J. Lu and K. Whitehouse, "Flash Flooding: Exploiting the Capture Effect for Rapid Flooding in Wireless Sensor Networks," in *IEEE INFOCOM 2009 - IEEE CONFERENCE ON COMPUTER COMMUNICATIONS, VOLS 1-5*, ser. IEEE INFOCOM, IEEE. IEEE, 2009, Proceedings Paper, pp. 2491–2499, IEEE INFOCOM Conference 2009, Rio de Janeiro, BRAZIL, APR 19-25, 2009.

[34] M. Maroti and J. Sallai, "Packet-level time synchronization," TinyOS Core Working Group, Technical Report, May 2008. [Online]. Available: http://www.tinyos.net/tinyos-2.x/doc/pdf/tep133.pdf

[35] 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.

[36] 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.

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

[38] K. S. Yildirim and A. Kantarci, "Drift estimation using pairwise slope with minimum variance in wireless sensor networks," *Ad Hoc Networks*, no. 0, pp. –, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870512001618

[39] 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 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. He is a member of IEEE.

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
DETAILS ON THE EXPERIMENTS

A.1 Hardware Platform

The hardware platform used for the implementation and experiments is MICAz from Memsic[7]. MICAz platform includes low-power 8-bit Atmel Atmega128L micro-controller which has 4 kB RAM and 128 kB program flash memory. The Chipcon CC2420 radio chip provides a 250 kbps data rate at 2.4 GHz frequency. 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 921 kHz (approximately 1 microsecond).

A.2 Implementation of the Protocols

We implemented EGSync and GTSP[8] in TinyOS 2.1.1 for our experiments. It is well-known that MAC layer timestamping reduces the effect of the non-deterministic delays on the message path and increases the quality of time synchronization [7], [1], [5], [14], [32]. Hence, in our implementations, we used CC2420 radio chip implementation of *packet level time synchronization* interfaces [34] provided by TinyOS for MAC layer timestamping. We used the latest implementation of FTSP coming with TinyOS 2.1.1 and modified it to work with a fixed reference node by disabling dynamic reference node election mechanism.

A.3 Testbed Setup

We used a testbed of 20 sensor nodes which are placed in the communication range of a *reference broadcaster* sensor node, as shown in Figure 14. We constructed line and ring topologies by configuring each node such that it will accept incoming messages from the nodes with identifier one more or one less than the identifier of the current node. In order to collect the logical clock values, the reference broadcaster transmits query packets periodically. The interval between these query packets is uniformly distributed between 20 and 23 seconds. Each query message is received approximately at the same time by all nodes. Then, the nodes broadcast the value of their logical clocks at the receipt time of the corresponding query packet. The base station attached to a PC transfers these messages to the serial port. An application listening the serial port logs these messages. At the end of the experiments, the evaluation metrics are applied to the collected data and the results are analyzed.

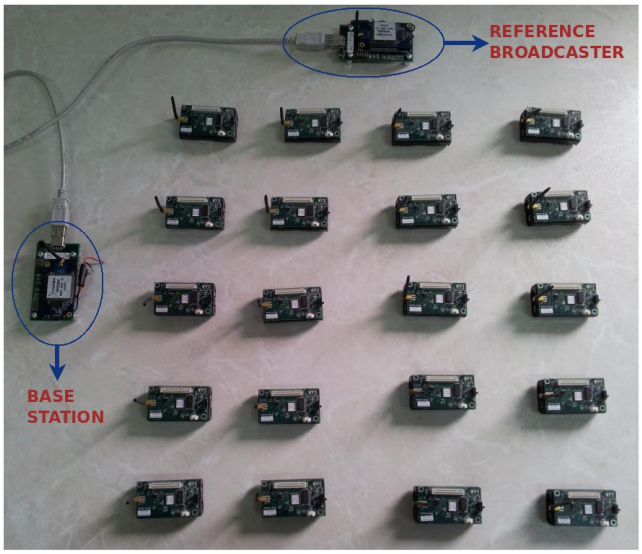

Figure 14: The testbed of 20 MICAz sensor nodes.

APPENDIX B
ANALYSIS OF EGSYNC PROTOCOL

In this section, we prove that the synchronization error of nodes executing EGSync is bounded. Through the execution of EGSync, the reference node floods its rate multiplier and time offset into the network while all of the other nodes including the reference node participate in the agreement algorithm. The following theorem states that if uncertainties through the message path are neglected, sensor nodes agree on a common logical clock speed by employing the clock speed update equation 6.

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

Proof: The equation 6 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

$$a_{ij} = \begin{cases} 1/|\mathcal{N}_i| + 1 & \{i,j\} \in E \\ 0 & otherwise \end{cases}.$$

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 [31], [14]. This result also implies that limiting the number of neighbors of a sensor node does not prevent agreement as long as the communication graph remains strongly connected. □

After the clock speed agreement is established, it can be proven that sensor nodes agree on a common logical clock value by employing the clock offset update equation 7.

Lemma B.2: If the graph G representing the communication network remains strongly connected, then $\forall v, u \in V : \lim_{t\to\infty} (L_v(t) - L_u(t)) = 0$.

Proof: It is also proven in [31], [14] that by employing the 7 equality at each synchronization message receipt, the logical clocks of the sensor nodes will converge to

7. http://www.memsic.com

8. Since GTSP does not have any publicly available implementation, we also implemented GTSP it in TinyOS ourselves.

a common value. The proof is based on establishing consensus by using distributed averaging. □

However, the uncertainties on the message path, i.e. *message delay*, directly effects the error of the synchronization in WSNs. Hence, message delay must be taken into account in order to obtain a realistic upper bound on the synchronization accuracy. In [35], [36], [1], message delay is modeled as a Gaussian random variable. By following these studies, we also model message delay as a normally distributed random variable with mean zero and variance σ^2.

As we mentioned, the slope of the estimated regression line is an estimate for the relative hardware clock rate. Hence, the probability distribution of the slope of the regression line h_v^u in the equality 6 which represents the speed of the hardware clock of node u with respect to that of the hardware clock of node v can be given as [37]:

$$h_v^u \sim \mathcal{N}\left(\frac{h_u}{h_v}, \sigma^2/S_{xx}\right). \quad (11)$$

where $S_{xx} = \sum(x_i - \overline{x})^2$ such that (x_i, Y_i) represents the collected data points in the regression table.[9] Assume that $\frac{1}{S_{xx}} \leq \frac{1}{\mathcal{S}}$ and $l_v \leq \ell$ hold for all times t and for all nodes $v \in V$.[10] Hence, h_v^u can be rewritten as

$$h_v^u = \frac{h_u}{h_v} + d_{uv} \quad (12)$$

such that $d_{uv} \sim \mathcal{N}(0, \frac{\sigma^2}{\mathcal{S}})$ represents the error of the regression due to the non-deterministic delays occurred during the communication between the nodes u and v. Hence,

$$\begin{aligned} h_v^u.l_v^u &= \frac{h_u}{h_v}.l_v^u + d_{uv}.l_v^u \\ &= \frac{h_u}{h_v}.l_v^u + \nu_{uv} \end{aligned} \quad (13)$$

holds such that $\nu_{uv} \sim (0, \frac{\sigma^2}{\mathcal{S}}\ell^2)$. By using this equality, the equality 6 can be rewritten as follows:

$$\begin{aligned} l_v(t^+) &= \frac{l_v(t) + \sum_{u\in\mathcal{N}_v} h_v^u(t).l_v^u(t)}{|\mathcal{N}_v| + 1} \\ &= l_v(t) + \frac{\sum_{u\in\mathcal{N}_v}(\frac{h_u}{h_v}.l_v^u(t) - l_v(t))}{|\mathcal{N}_v| + 1} \\ &\quad + \frac{\sum_{u\in\mathcal{N}_v}\nu_{uv}(t)}{|\mathcal{N}_v| + 1} \\ &= l_v(t) + c\sum_{u\in\mathcal{N}_v}(\frac{h_u}{h_v}.l_v^u(t) - l_v(t)) + n_v(t) \end{aligned} \quad (14)$$

where $c = 1/(|\mathcal{N}_v| + 1)$ and $n_v(t) = c\sum_{u\in\mathcal{N}_v}\nu_{uv}(t)$.

9. Please see [38] for the details.

10. 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.

It can be shown that the equality 14 is equivalent to the equality 5 in [39].[11] Hence, the following theorem holds:

Theorem B.1: If the graph G remains strongly connected, then the variance of l_v for any $v \in V$ is upper bounded by

$$c\frac{\sigma^2}{\mathcal{S}}\ell^2 min\{\mathcal{D}_n max\{\lambda_i\}, \lambda_{max}(A^2)\sum_{i=2}^n \lambda_i\}, \quad (15)$$

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 [39] with $\mathbf{u}= 0$, $\varepsilon = c$ and $\sigma^2 = \frac{\sigma^2}{\mathcal{S}}\ell^2$. □

It can also be shown that the clock offset update equation 7 is also equivalent to the equality 5 in [39]. Consequently, we conclude that the errors of the clock speed and clock value agreement algorithms employed by EGSync essentially depend on the total degree of the network and the eigenvalues of L and A^2. Therefore, the difference of the logical clock values of the reference node ref and node v, i.e. $L_v - L_{ref}$, is bounded.

Now consider the offset propagation mechanism of EGSync. After the clock value and speed agreement is achieved, the reference node propagates $\Delta_{ref} = H_{ref} - L_{ref}$ through the network periodically. When any node v receives this difference, it can estimate the value of the hardware clock of the reference node by adding the received difference to L_v. The error of this estimation is bounded for all times after the agreement is achieved since the error of $L_v - L_{ref}$ is bounded.

APPENDIX C
SYNCHRONIZATION TO THE REAL-TIME

In general, external synchronization in WSNs can be achieved by applying the following steps. First, a reference node is synchronized to an external time source, such as UTC time via a GPS receiver. Second, the remaining sensor nodes are synchronized to the reference node by employing internal synchronization. If the reference node is equipped with a GPS receiver and has access to the 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 speed, it can estimate its hardware clock speed at time t such that $t_0 < t_1 \leq t$ as

$$\hat{h}_{ref}(t) = \frac{H_{ref}(t_1) - H_{ref}(t_0)}{t_1 - t_0}. \quad (16)$$

Using this value, the reference node can also estimate the common logical clock speed inside the network after the clock speed agreement as follows

11. Consider equality 4 in [39]. 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 in [39].

$$\hat{speed} = \hat{h}_{ref}(t).l_{ref}(t) \quad (17)$$

since it knows $l_{ref}(t)$ and it can estimate $h_{ref}(t)$ by using equality 16. The reference node is now required to flood synchronization messages into the network which carry $\hat{speed}$ instead of $l_{ref}(t)$ and $\Delta_{ref}(t) = t - L_{ref}(t)$ instead of $\Delta_{ref}(t) = H_{ref}(t) - L_{ref}(t)$.

As a final modification, each sensor node v is required to calculate its logical clock using the following equation instead of the equation 9:

$$L_v(t) = \int_0^t h_v(\tau) \frac{l_v(\tau)}{\hat{speed}} d\tau + \theta_v(t). \quad (18)$$

Thus, node v progresses its logical clock at the speed of the real-time, i.e. 1, and it is able to output the real-time at any time by adding Δ_v^{ref} to L_v.