2010 Second International Conference on Computational Intelligence, Communication Systems and Networks

Energy-Efficient Gradient Time Synchronization for Wireless Sensor Networks

Kittipat Apicharttrisorn, Supasate Choochaisri, Chalermek Intanagonwiwat
Department of Computer Engineering, Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
kittipat.ap@student.chula.ac.th, supasate.c@student.chula.ac.th, chalermek.i@chula.ac.th

***Abstract*—Wireless sensor network (WSN) applications usually demand a time-synchronization protocol for node coordination and data interpretation. In this paper, we propose an Energy-Efficient Gradient Time Synchronization Protocol (EGTSP) for Wireless Sensor Networks. In contrast to FTSP, a state-of-the-art synchronization protocol for WSNs, EGTSP is a completely localized algorithm that achieves a global time consensus and gradient time property using effective drift compensation and incremental averaging estimation. In contrast with GTSP, a gradient-based fixed-rated time synchronization protocol, our protocol provides adaptive beaconing for applications to optimize energy savings by selecting appropriate message-broadcast periods. The protocol is implemented and evaluated on multi-hop networks that consist of Telosb motes running TinyOS. The experimental results indicate that our protocol achieves a network-wide global notion of time, attains small synchronization errors, and utilizes energy efficiently.**

***Keywords*-time synchronization; wireless sensor networks; energy efficiency**

I. Introduction

Wireless sensor networks (WSNs) have been extensively explored by research communities worldwide for a decade. Small-sized, resource-constraint, and battery-powered sensor nodes are wirelessly connected to each other to disseminate sensed data to sinks. Given that energy is one of the most precious resources in WSNs, it is mandatory that protocols in WSNs are energy-efficient. WSNs are unprecedentedly capable of monitoring environments which are sensitive to human access, infrastructure which is hard to maintain, and patients who are annoyed by electrical wires. Sensor network applications usually require that time and space information of an event be embedded into sensed data so that the data are more meaningful and usable. For all sensor nodes in the network to have a common notion of time, time synchronization is needed.

However, time synchronization is classified into three models proposed by Ganeriwal et al. [1]: event ordering, relative clock and global clock (or always-on). An event-ordering model is the simplest model whereby nodes in the system maintain only a right order of events. An example of this model is a protocol by Romer[2]. In a relative clock model, a node in sensor networks only maintains its clock relations between itself and neighbors. Example protocols of this model are RBS[3], and Tiny-Sync[4]. A global clock model (the most complex but most useful one) requires that all nodes maintain a common notion of time (the global time). With synchronized global time , nodes can easily have a right order of events, and a right relation of time with other nodes. Additionally, nodes can perform TDMA or schedule their duty cycle with minimal wake-up guard time.

Global time synchronization is a classical problem in distributed systems [5]. Network Time Protocol (NTP) [6] on Internet has provided global time synchronization for decades but, because of its high complexity and low accuracy in the millisecond scale, NTP is not well suited for use in WSNs. A state-of-the-art global time-synchronization protocol for WSNs is Flooding Time Synchronization Protocol (FTSP) [7].

However, FTSP lacks a gradient time property or locality. Most WSN applications require that nearer nodes have tighter time synchronization than farther nodes do. For example, most object tracking applications require that nodes adjacent to each other have tighter time synchronization to correctly detect the direction and speed of an object. Conversely, nodes with multi-hop away can tolerate larger synchronization error.

Unlike FTSP, Gradient Time Synchronization Protocol (GTSP) [8] is a distributed time synchronization protocol with the gradient time property. In GTSP, each node estimates its perceived global (logical) time with the average perceived global time of all neighbors. Every node periodically broadcasts a reference-point message containing its global time. The received reference points from all neighbors are used to calculate an average time offset and relative global rate. According to this computed time information, the global clock of the computing node is adjusted.

In addition, the global clock of each node is adjusted once every broadcasting period. Consequently, the amount of adjustment can be large. Therefore, the global time will not be smoothed (continuous). This may cause an error in data interpretation of an application. Furthermore, in GTSP, the broadcasting period is fixed. To achieve small synchronization error, this fixed broadcasting period of GTSP must also be small. This will linearly and significantly drain sensor networks' energy.

In this paper, we propose an Energy-Efficient Gradient Time Synchronization Protocol (EGTSP) for wireless sensor networks. Similar to GTSP, EGTSP is a localized algorithm to achieve the global time consensus and the gradient time property. However, unlike GTSP, EGTSP achieves those properties using effective drift compensation and incremental averaging estimation without the mentioned drawbacks of GTSP.

To improve the continuity of the global time, each EGTSP node computes the incremental average of time immediately after receiving a broadcast message from a neighbor (instead of computing the average after broadcast messages from all neighbors are received). Given that this incremental adjustment is smaller, the continuity of the global time is improved.

Additionally, after the drift estimation, our algorithm can adaptively adjust the broadcasting period to save energy without degrading the global time accuracy. Applications are also allowed to select an extensive broadcasting period by trading off metrics such as synchronization error, system responsiveness, and energy consumption.

We have also implemented and evaluated our protocol on a testbed consisting of Telosb motes, each of which runs TinyOS. EGTSP is compared with GTSP to verify the viability of our approach. The experimental results indicate that EGTSP consumes 82% less energy as well as incurs slightly smaller synchronization error over GTSP does.

II. Related Work

Time synchronization techniques can be simply divided into two categories: receiver-to-receiver and sender-to-receiver. In a receiver-to-receiver protocol, one node broadcasts a beacon message to let all nodes in the broadcast domain timestamp upon reception. The receivers of the broadcast later exchange timestamp information to each other and calculate time offset relative to each other. Using this technique, the effect of delay uncertainties at sender side is minimized; thus, the accuracy of time synchronization is improved. An Example of receiver-to-receiver protocols includes Reference Broadcast Synchronization (RBS) [3].

In a sender-to-receiver protocol, a sender stamps its sending time into its broadcasting messages. A receiver of the message also records its receiving time and then calculates the time offset relative to the sender. The receiver can use this time offset to adjust its clock accordingly. In Timing-sync Protocol for Sensor Networks (TPSN) [1], Ganeriwal et al have mathematically proved that, by time-stamping messages in the MAC layer, a sender-to-receiver protocol achieves better synchronization accuracy than a receiver-to-receiver protocol does. However, TPSN depends time synchronization on a tree structure in which a child synchronizes to only its parent node. Consequently, TPSN is vulnerable to network dynamics, which potentially occur in WSNs. Our proposed protocol (EGTSP) also uses the sender-to-receiver synchronization but does not construct any tree of time relationships. EGTSP relies on time information from all neighbors rather than one. As a result, our distributed and localized protocol is robust to network dynamics.

To solve the mentioned problem of TPSN, FTSP [7] floods time-synchronization messages from an elected root to provide redundant messages from several routes. However, FTSP's performance depends heavily on the root. Any change on the root (e.g., root failure, root position in the network, root re-election) will significantly degrade the performance. Furthermore, flooding techniques are notorious in draining network energy. Extending flooding periods will consequentially reduce synchronization accuracy. Furthermore, FTSP lacks the gradient time property that is desirable to most WSN applications.

Conversely, our proposed protocol does not depend on any root node. Every node equally participates in synchronization processes. Therefore, EGTSP is robust to a single point of failure. Additionally, with adaptive beaconing, EGTSP is also energy-efficient. At the beginning, nodes broadcast messages at a fixed period to compensate for time offsets and concurrently evaluate clock drifts. After the evaluation completes, nodes can extend a synchronization period, saving energy in broadcasting. In addition, given that EGTSP nodes compute their global time from all neighbors' time, EGTSP does not lack the gradient property.

Of a particular interest is Elapsed Time on Arrival (ETA) from Kusy et al. [9]. ETA provides the source-to-destination traveling time (elapsed time) of a message. Based on this elapsed time, the destination node can compute the relative time between the source and the destination. Understandably, this relative time is a basis to build a time synchronization protocol. However, ETA alone does not provide the global notion of time whereas our protocol does. Similarly, Tiny-sync [4] has been proposed to synchronize time between a pair of nodes. However, unlike our approach, Tiny-sync does not provide a mechanism to efficiently synchronize the time for the whole network.

In contrast to Tiny-sync, Time-diffusion [10] provides a converged global time of a sensor network. Nevertheless, each node must be a member of several trees as well as synchronize time with an elected diffused leader, causing high complexity in terms of overhead and memory. Conversely, the overhead and memory usage of our work are quite small.

To minimize overhead, Asynchronous Diffusion [11] requires only that each node participates in synchronization processes with a non-zero probability. It is proved that the time of an entire network can eventually converge to the global time. Each operation of Asynchronous Diffusion requires three-way message handshakes whereas each operation of our algorithm requires only one message broadcast. Hence, Asynchronous Diffusion potentially incurs more overhead than our work does when the participation probability is higher than 0.33. With a lower probability, the overhead should be reduced but the converging time may be adversely affected.

III. Energy-Efficient Gradient Time Synchronization Protocol

A wireless sensor network can be simply modeled as a connected graph, $C = (V, E)$, whereby V is a set of nodes and E is a set of communication links between nodes. Each sensor node is equipped with a local clock, L_i , that is initialized to Θ_i. The local clock progresses in accordance with its hardware clock tick h_i. Even if all nodes start with the same clock value, the local clocks of the nodes will drift apart from each other as time passes, called clock drifts. Therefore, nodes need to periodically exchange time information in order to maintain an acceptable level of synchronization error. The drift rate of each sensor node depends on temperature, battery power, and oscillator age. The clock model is defined as

$$L_i(t) = \int_{t_0}^{t} h_i(\tau)\, d\tau + \Theta_i(t_0) . \qquad (1)$$

EGTSP estimates the global time and the average neighbor clock drift. The global time is the local time compensated by the average time offset (relative to the neighbors' perceived global time) and the average drift. If a node has the perceived global time higher than the average of the neighbors' global time, it will decrease its global time. Conversely, if a node has the global time lower than the average of the neighbors' global time, it will increase its global time accordingly. By repeating this process, the clocks of the sensor nodes in a network will converge to a common value, which is the global clock. Subsequently, the average neighbor clock drift is estimated using local clocks of the neighbors (relative to the computing node's local clock). If the average drift is more than 1 (the below line in Fig. 1), the clock of the computing node is running more slowly than that of its neighbors. Therefore, the node has to increase its global clock. If the average drift is less than 1 (the above line in Fig. 1), the clock of a node is running faster than that of its neighbors. Thus, the node has to decrease its global clock.

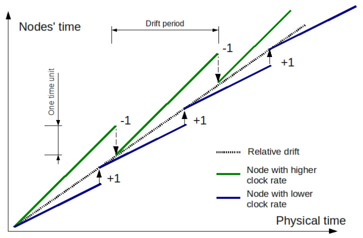

Figure 1. Neighborhood drift compensation

The inverse of an average drift indicates the amount of time for the node's perceived global clock to drift one time unit apart from the neighbors' global clock. This time amount is called a *drift period*. At the end of every drift period, nodes increase or decrease their global clocks one time unit. The benefits of the average-drift estimation are twofold.

1) Once the global clock is converged, the correctly estimated drift alone is sufficient to maintain the global time. Therefore, without network dynamics, nodes do not have to broadcast synchronization messages at all. Even if they do, the broadcast period can be extended. This will save energy in overhead significantly.
2) Sensor nodes can sleep and turn off the radio for a longer period to save even more energy, because they no longer have to frequently exchange the synchronization messages with each other. To maintain the global time after waking up, the node simply calculates how long it has slept and estimates how many time units it has to compensate for the drift.

In this paper, the global clock is modeled as

$$G_i(t) = L_i(t) + \theta_i(t) + \omega_i(t) \qquad (2)$$

where $G_i(t)$is the global time of node i at time t, $L_i(t)$ is the local time and $\theta_i(t)$ is an offset compensation value

$$\theta_i(t) = \theta_i(t-1) + \frac{G_j(t-1) - G_i(t-1) + \gamma_i(t-1)}{|N_i| + 1} \qquad (3)$$

where N_i is a neighbor set of node i and node $j \in N_i$, and $\gamma_i(t-1)$ is the remainder of the previous integer division. The remainder of an iteration is calculated by modular operation. $\omega_i(t)$ is a drift compensation value, defined as

$$\omega_i(t) = \begin{cases} \omega_i(t-1) + 1 & \text{every positive drift period} \\ \omega_i(t-1) - 1 & \text{every negative drift period} \\ \omega_i(t-1) & \text{else} \end{cases} \qquad (4)$$

EGTSP's table structure, message structure, and algorithms can be described in Table I, Fig. 2 and, Alg. 1 and 2, respectively.

Table Field	Description
NodeID	A neighbor's node ID
SeqNum	A sequence number of a broadcast from a neighbor
SendLocalTime	Local sending time of the sender
RcvLocalTime	Local receiving time of the receiver

Table I
Table structure

Alg 1. starts by statically declaring the maximum number of entries of the table. This number should not be smaller than the maximum degree of nodes in the network. Upon

NodeID	SeqNum	SendLocalTime	SendGlobalTime

Figure 2. Message structure

Algorithm 1 Offset compensation

```
1: TableItem table[MAX_ENTRIES]
2: Node i receives a time sync msg from node j
3: Timestamp the local receiving time, $L_i$ and also the global receiving time $G_i$ using (2)
4: Sending node's time, $L_j = msg->SendLocalTime$, $G_j = msg->SendGlobalTime$
5: Update table, table.NodeID = j , $table.SeqNum = msg->SeqNum$, table.SendLocalTime = $L_j$ , table.RcvLocalTime = $L_i$
6: Calculate time offset between node i and j, $\theta_{i,j} = G_j - G_i$
7: if $\theta_{i,j} > JUMP\_THRESHOLD$ then
8:     $G_i = G_i + \theta_{i,j}$
9: else
10:    $G_i = G_i + \theta_{i,j}/(N_i + 1)$ // incremental averaging, see (3) for details
11: end if
12: When the sync timer expires, node i broadcasts a time sync msg and resets the sync timer
```

Algorithm 2 Drift Compensation

```
1: TableItem startCapture[MAX_ENTRIES], stopCapture[MAX_ENTRIES];
2: Allow a period for nodes to collect initial reference points
3: Start capturing the drift by duplicating table to startCapture
4: For the CAPTURE_PERIOD, allowing newcoming reference points to update table
5: Stop capturing the drift by duplicating an updated table to stopCapture
6: Calculate relative drift by $D_i = \frac{\left(\sum_{j \in N_i} \frac{L_j^{(stop)} - L_j^{(start)}}{L_i^{(stop)} - L_i^{(start)}} + 1\right)}{N_i + 1}$
7: Calculate drift period, $DP_i = \frac{1}{(D_i - 1)}$
8: if $DP_i > 0$ then
9:     for every $|DP_i|$ period, $\omega_i = \omega_i + 1$
10: else if $DP_i < 0$ then
11:    for every $|DP_i|$ period, $\omega_i = \omega_i - 1$
12: end if
13: If the drift is converged, inform the application to choose a longer time-sync period.
```

receiving a message, node i timestamps both the global clock and the local clock. The global reception time is used to calculate the relative offset while the local reception time is used to further calculate the drift. If the relative offset is positively over a threshold, the receiver's global time will be adjusted to the sender's global time. Otherwise, the receiver's global time is added with a fraction of the relative offset. In a sense, the global time is incrementally averaged.

Alg 2. starts by declaring two more tables of the table structure in Table 1. The two tables are used to capture the relative neighbor drift for an extended period (CAPTURE_PERIOD). The longer period to estimate drifts, the less probability of estimation error [12]. In contrast to GTSP whereby a fixed period is used for both offset estimation and drift estimation, we argue that the periods for both estimations should be treated differently. A period for effective offset compensation should be shorter than the time-sync period with incremental averaging for smoother global time. Conversely, the period for effective drift compensation must be long enough to avoid estimation error but be short enough for the system to quickly start taking advantage of the drift estimation. For example, D_i of 1.000001 indicates that node i's clock drifts below the neighbor drift. Therefore, DP_i = 1/(1.000001 - 1) = 10^6 . If the precision of the hardware clock is millisecond, the node will increase the global time one millisecond every 16.67 minutes. As a result, WSN applications on this node can extend the time-sync period and the sleep-wake-up period as long as the system responsiveness to network dynamics and other metrics are still acceptable. This allows the WSN applications to optimize energy management, more flexibly and effectively than using a fixed beacon rate as proposed by other work.

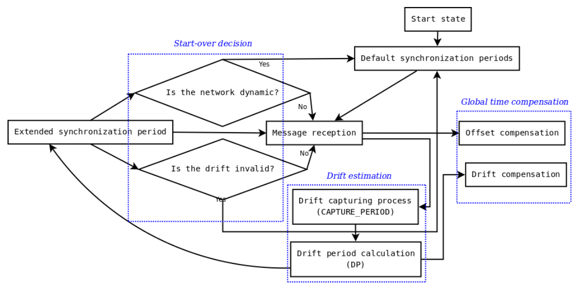

Figure 3. Algorithm Diagram

From Fig. 3, the algorithms will start over if the network dynamic is detected or the drift estimation is no longer accurate. Network dynamics include newly joining nodes, leaving nodes, or moving nodes. Furthermore, we need to detect if the neighbor drift estimation has become invalid. From Fig 1, one may observe that if sensor nodes' clocks are still compensating for a valid relative drift, the offset between any two nodes will not exceed two time unit. To enhance the confidence of the observation, nodes may count the consecutive times when this situation occurs.

Time synchronization messages are not only used to attain offset compensation, but also used to detect network dynamics and invalid relative drift. WSN applications will have to trade off system responsiveness (to network dynamics) with energy consumption. However, we argue that our time synchronization protocol consumes significantly less energy, poses considerably less overhead, and maintains acceptably small synchronization errors.

IV. Experimental Results

Our experiments are conduct on Telosb sensor nodes [13] running TinyOS 2.1. Our experiments consist of two scenarios: one-hop and multi-hop. Our multi-hop networking is software controlled. Specifically, the received packets will be dropped if the packets are sent from nodes that are not neighbors in the logical topology (even though they may be physically neighbors).

Variable	Value
Number of nodes	7
Default sync period	30 s
Global time sampling period	30 s
Experiment period (approx)	~10000 s
Drift capture period (of EGTSP)	300 s

Table II
Experimental settings

Algorithms	Total messages (bytes)
GTSP	54432
EGTSP	9856

Table III
Message overhead and energy consumption : Total time synchronization messages (bytes)

Algorithms	Single-hop	Multi-hop
GTSP	0.52	1.52
EGTSP	0.52	1.45

Table IV
Accuracy comparison: average synchronization errors (milliseconds)

EGTSP poses significantly less synchronization-message overhead than GTSP does in our experiment (see Table III). The main reason is that EGTSP can extend the synchronization period once the neighborhood drift is successfully estimated without degrading the synchronization error but GTSP cannot. If GTSP extends its synchronization period, the synchronization error of GTSP will increase (see below). Specifically, EGTSP can achieve 82% energy savings under investigated scenarios given that GTSP sends 5.5 times of the total synchronization messages sent by EGTSP.

In Fig. 4 and 5, our experimental results indicate that network time synchronization errors of EGTSP are comparable to those of GTSP in a single-hop network. Network synchronization errors are computed by averaging differences of perceived global time from all nodes in the network

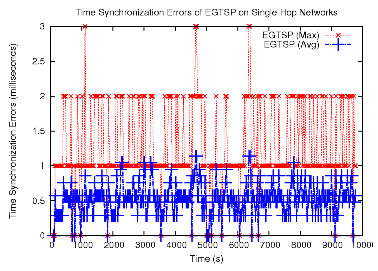

Figure 4. EGTSP's time synchronization errors on single-hop networks

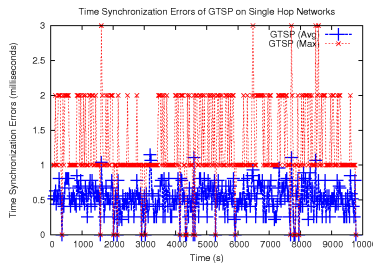

Figure 5. GTSP's time synchronization errors on single-hop networks

whereas neighbor synchronization errors are computed using only global time differences between neighbors. In a single-hop network, neighbor synchronization errors and network synchronization errors are equal.

In Fig. 6, EGTSP's neighbor synchronization errors (below) are smaller than network synchronization errors (above) in all samplings of the experiment. This result indicates that EGTSP also holds the gradient property of GTSP. In Fig. 7, ineffective drift estimation of GTSP causes synchronization errors to accumulate in the middle of the experiment. Because of its frequent synchronization messages, GTSP's synchronization errors are pulled back to a lower level again. Therefore, if the synchronization period of GTSP is extended, the duration of the higher-error level will be as well extended. Conversely, in Fig. 6, EGTSP is more stable than GTSP due to its incremental averaging (which adjusts time offset upon reception of a message) and its effective drift estimation (which is carefully used for drift compensation of the global time). As a result, our protocol is slightly more accurate but significantly more energy-efficient than GTSP (Table III and IV).

V. Conclusions

In this paper, we propose EGTSP, an energy-efficient gradient time synchronization protocol for providing a global notion of time in wireless sensor networks. EGTSP is distributed, gradient-based, and energy-efficiency-oriented. Therefore, we introduce a concept of neighborhood drifts for effective drift compensation and adaptive adjustment of broadcasting periods to save energy. EGTSP also includes frequent offset compensation with incremental averaging to

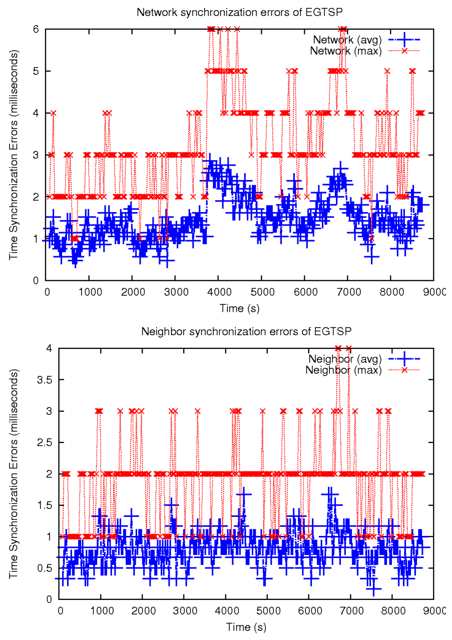

Figure 6. EGTSP's synchronization errors on multi-hop networks between all nodes (above) and between neighbors (below)

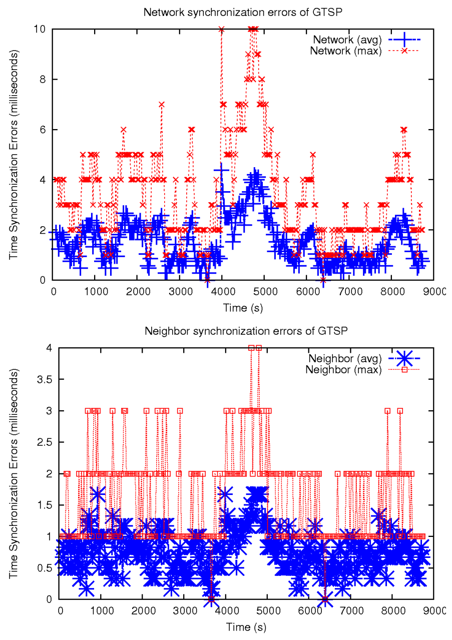

Figure 7. GTSP's synchronization errors on multihop networks between all nodes (above) and between neighbors (below)

smooth the global time estimation. We evaluate our protocol on Telosb sensor motes and compare the performance of EGTSP with that of GTSP. The experimental results indicate that, under investigated scenarios, our synchronization protocol is much more energy-efficient than GTSP is, without degrading the time synchronization accuracy. Future work includes improving an accuracy of clock drift evaluation and simulating EGTSP on a more scalable topology.

References

[1] 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*. ACM, 2003.

[2] K. Römer, "Time synchronization in ad hoc networks," in *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*. ACM, 2001.

[3] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," in *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*. ACM, 2002.

[4] S. Yoon, C. Veerarittiphan, and M. L. Sichitiu, "Tiny-sync: Tight time synchronization for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 3, no. 2, p. 8, 2007.

[5] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[6] D. L. Mills, "Internet time synchronization: The network time protocol," 1989.

[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*. ACM, 2004.

[8] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*. IEEE Computer Society, 2009.

[9] 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.

[10] W. Su and I. F. Akyildiz, "Time-diffusion synchronization protocol for wireless sensor networks," *IEEE/ACM Trans. Netw.*, vol. 13, no. 2, pp. 384–397, 2005.

[11] Q. Li and D. Rus, "Global clock synchronization in sensor networks," *IEEE Trans. Comput.*, vol. 55, no. 2, pp. 214–226, 2006.

[12] C. Lenzen, P. Sommer, and R. Wattenhofer, "Optimal clock synchronization in networks," in *SenSys '09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2009.

[13] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*. IEEE Press, 2005.