Wireless Sensor Networks: A Networking Perspective, Edited by [Jun Zhang and Abbas Jamalipour].
ISBN 0-471-XXXXX-X Copyright © 2000 Wiley[Imprint], Inc.

Chapter 9

Time Synchronization

Fikret Sivrikaya and Bülent Yener

Rensselaer Polytechnic Institute, USA

9.1 Introduction

Time synchronization, as in all distributed systems, is an important component of a wireless sensor network (WSN), which aims to provide a common timescale for local clocks of nodes in the network. Since all hardware clocks are imperfect, those at different nodes may drift away from each other in time. For this reason, the observed time or durations of time intervals may differ for each node in the network. However, for many applications or networking protocols, it is required that a common view of time exists and is available to all or some of the nodes in the network at any particular instant.

In this chapter, we focus on the time synchronization problem, and review existing synchronization methods and protocols for WSNs. In Section 9.1, we introduce the synchronization problem and the common challenges for synchronization methods. In Sections 9.2 and 9.3, we discuss the need for synchronization and the requirements of synchronization in WSNs, respectively. In Section 9.4, we review existing synchronization methods and protocols for WSNs. In Section 9.5, we conclude with a summary of this chapter and a brief discussion of future research directions.

9.1.1 Computer Clocks and the Synchronization Problem

Computing devices are mostly equipped with a hardware oscillator-assisted computer clock, which implements an approximation $C(t)$ of real time t as

$$C(t) = k\int_{t_0}^{t} \omega(\tau)d\tau + C(t_0),$$

where $\omega(t)$ is the angular frequency of the hardware oscillator, k is a proportionality coefficient, and t_0 is the initial value of the clock [1]. For a perfect clock, dC/dt would equal 1. However, all clocks are subject to a clock drift; the oscillator frequency will vary unpredictably due to various physical effects. Even though the frequency of a clock changes over time, it can be approximated with good accuracy by an oscillator with a fixed frequency [2]. Then for some node i in the network, we can approximate its local clock as

$$C_i(t) = a_i t + b_i,$$

where a_i is the clock drift, and b_i is the offset of node i's clock. *Drift* denotes the rate (frequency) of the clock, and *offset* is the difference in value from real time, t. Using the equation above, we can compare the local clocks of two nodes in a network, say, nodes 1 and 2 as

$$C_1(t) = a_{12} \cdot C_2(t) + b_{12} . \quad (1)$$

We call a_{12} the *relative drift*, and b_{12} the *relative offset* between the clocks of node 1 and node 2. If two clocks are perfectly synchronized, their relative drift is 1, meaning that the clocks have the same rate, and their relative offset is zero, meaning that they have the same value at that instant. Some studies in the literature use “skew” instead of drift, defining it as the *difference* (as opposed to *ratio*) between clock rates [3-4]. Also, offset may equivalently be mentioned as “phase offset”.

The synchronization problem on a network of *n* devices corresponds to the problem of equalizing the computer clocks of different devices. The synchronization can be either *global*, trying to equalize $C_i(t)$ for all $i=1, 2, \ldots, n$, or *local*, trying to equalize $C_i(t)$ for some set of the nodes – mostly those that are spatially close or on the same path between communicating nodes. Equalizing just the instantaneous values (correcting the offsets) of clocks is not enough for synchronization because the clocks will drift away afterward. Therefore, a synchronization scheme should either equalize the clock rates as well as offsets, or repeatedly correct the offsets to keep the clocks synchronized over a time period.

The above definition of synchronization actually outlines the strictest form of synchronization, where one seeks perfect matching of time on different clocks, but this definition can be relaxed to different degrees, according to the need of an application. In general, the synchronization problem can be classified into three basic types [5]. The first and simplest type of synchronization deals only with the ordering of events or messages. The aim of such synchronization is to tell whether an event E_1 has occurred before or after another event E_2 (i.e., just compare the local clocks for order rather than have them synchronized). The synchronization protocol proposed in [6] is an example of this type. The second type of synchronization targets maintaining relative clocks. In such synchronization, a node runs its local clock independently, but keeps information about the relative drift and offset of its clock to other clocks in the network so that at any instant the local time of the node can be converted to some other node's local time and vice versa. Most of the synchronization protocols proposed for sensor networks use this model [2-3][7]. The third and most complex type of synchronization is the “always on” model, where all nodes maintain a clock that is synchronized to a reference clock in the network. The goal of this type of synchronization is to preserve a global timescale throughout the network. The synchronization protocol proposed in [5] conforms to this model, but the use of “always on” model is not mandatory in the protocol.

9.1.2 Common Challenges for Synchronization Methods

All network time synchronization methods rely on some sort of message exchange between nodes. Non-determinism in the network dynamics such as propagation time or physical channel access time makes the synchronization task challenging in many systems. When a node in the network generates a timestamp to send to another node for synchronization, the packet carrying the timestamp will face a variable delay until it reaches and is decoded at its intended receiver. This delay prevents the receiver from exactly comparing the local clocks of the two nodes and accurately synchronizing to the sender. We can basically decompose the sources of errors in network time synchronization methods into four basic components:

- *Send Time*: This is the time spent to construct a message at the sender. It includes the overhead of operating system (e.g., context switching) and the time to transfer the message to the network interface for transmission.
- *Access Time*: Each packet faces some delay at the medium access control (MAC) layer before actual transmission. The sources of this delay depend on the MAC scheme used, but some typical reasons for delay are waiting for the channel to be idle or waiting for the time-division multiple access (TDMA) slot for transmission.

- *Propagation Time*: This is the time spent in propagation of the message between the network interfaces of the sender and the receiver.
- *Receive Time*: This is the time needed for the network interface of the receiver to receive the message and transfer it to the host.

In large networks, the propagation time may become quite large and important because it includes the queuing and switching delays at the routers on a path between two nodes. However, for two nodes in a sensor network within the transmission range of each other, this delay is just the propagation time of a packet in the air, which is typically very small. In the Mica hardware platform, the sensor node architecture of Berkeley [8], a system-level optimization for wireless sensor architecture is proposed, which can be used for removing the effect of delay caused by *send time* and *access time* on the synchronization accuracy: If there is a tight coupling between the application and its communication protocol, the MAC layer can inform the application what delay a packet experiences before it is transmitted. This information can even be used to modify the packet once the transmission begins so that the timestamp in the packet reflects the exact time when it was sent [8]. Similarly, if the arrival time can be timestamped at an enough low level at the receiver, the error due to *receive time* can be decreased because it would not include operating system overheads, or the time to transfer the message from the network interface to the host [3].

9.2 Need for Synchronization in Wireless Sensor Networks

There are several reasons for addressing the synchronization problem in WSNs. First, sensor nodes need to coordinate their operations and collaborate to achieve a complex sensing task. Data fusion is an example of such coordination in which data collected at different nodes are aggregated into a meaningful result. For example, in a vehicle tracking application, sensor nodes report the location and time at which they sense the vehicle to a sink node, which in turn combines this information to estimate the location and velocity of the vehicle. Clearly, if the sensor nodes lack a common timescale (i.e., are not synchronized), the estimate will be inaccurate.

Second, synchronization can be used by power-saving schemes to increase network lifetime. For example, sensors may *sleep* (go into a power-saving mode by turning off their sensors and/or transceivers) at appropriate times and wake up when necessary. When using the power-saving mode, the nodes should sleep and wake-up at coordinated times such that the radio receiver of a node is not turned off when there is some data directed to it. This requires precise timing between sensor nodes.

Scheduling algorithms such as TDMA can be used to share the transmission medium in the time domain to eliminate transmission collisions and conserve energy. Thus, synchronization is an essential part of transmission scheduling.

Traditional synchronization schemes such as Network Time Protocol (NTP) [9] or Global Positioning System (GPS) [10] are not suitable for use in sensor networks because of complexity and energy issues, cost and size factors. NTP works well for synchronizing the computers on the Internet, but is not designed with the energy and computation limitations of sensor nodes in mind. A GPS device may be too expensive to be attached on cheap sensor devices, and GPS service may not be available everywhere (e.g., inside buildings or under water).

9.3 Requirements of Synchronization in Wireless Sensor Networks

In this section we present a broad set of requirements for the synchronization problem. These requirements can also be regarded as the metrics for evaluating synchronization schemes for

WSNs. However, there are trade-offs between the requirements on an efficient synchronization solution (e.g., *precision* vs. *energy efficiency*). A single scheme may not satisfy them altogether.

- *Energy Efficiency*: As with all protocols designed for sensor networks, synchronization schemes should take into account the limited energy resources in sensor nodes.
- *Scalability*: Most sensor network applications need deployment of a large number of sensor nodes. A synchronization scheme should scale well with increasing number of nodes and/or high density in the network.
- *Precision*: The need for precision, or accuracy, may vary significantly depending on a specific application and the purpose of synchronization. For some applications, even a simple ordering of events and messages may suffice, whereas for some others the requirement for synchronization accuracy may be on the order of a few microseconds.
- *Robustness:* A sensor network is typically left unattended for a long time of operation in possibly hostile environments. In case of the failure of a few sensor nodes, the synchronization scheme should remain valid and functional for the rest of the network.
- *Lifetime*: The synchronized time among sensor nodes provided by a synchronization algorithm may be instantaneous, or may last as long as the operation time of the network. If the synchronization scheme synchronizes the drifts and removes the offsets, the lifetime for the synchronized time is typically much higher.
- *Scope*: The synchronization scheme may provide a global time base for all nodes in the network or local synchronization only among spatially close nodes. Because of scalability issues, global synchronization is difficult to achieve or too costly (considering energy and bandwidth usage) in large sensor networks. On the other hand, a common time base for a large number of nodes might be needed to aggregate data collected from distant nodes, dictating a global synchronization.
- *Cost and Size*: Wireless sensor nodes are very small and inexpensive devices. Therefore, as noted earlier, attaching relatively large or expensive hardware (e.g., a GPS receiver) on a small cheap device is not a logical option for synchronizing sensor nodes. A synchronization method for sensor networks should be developed with limited cost and size in mind.
- *Immediacy*: Some sensor network applications such as emergency detection (e.g., gas leak detection, intruder detection) require the occurring event to be communicated immediately to the sink node. In such applications, the network cannot tolerate any kind of delay when such an emergency is detected. This is called the *immediacy* requirement, and may prevent a protocol designer from relying on excessive processing after such an event of interest occurs.

9.4 Synchronization Protocols for Wireless Sensor Networks

There has been such a significant amount of research on the time synchronization problem in WSNs that it would be impractical to present the details of all here. Instead, we identify three primary tracks for studying synchronization methods, and present some existing protocols as representatives for each track. The first one is the *synchronization primitives*, covering the methods for establishing instantaneous synchronization between neighbor nodes. In a long-lived multi-hop sensor network, the synchronization primitives are typically used as the building blocks for achieving network-wide and/or long-term synchronization. Hence the second and third tracks consider the *multi-hop synchronization* and *long-term synchronization*, respectively. At the end of the section, we summarize and comment on some other protocols and relevant work.

9.4.1 Synchronization Primitives

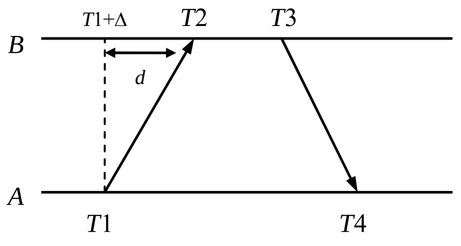
In this section, we focus on the methods for providing instantaneous synchronization between the local clocks of neighbor nodes in a sensor network. We classify these methods as synchronization primitives because they are mostly used as the basic building blocks for synchronizing nodes distributed throughout the network.

9.4.1.1 Two-Way Message Exchange

Two-way message exchange between a pair of nodes is the conventional method of synchronizing local clocks in a network, which is employed by NTP for traditional wired networks. This method is also the basic building block of many network-wide synchronization protocols for sensor networks, such as the Timing-sync Protocol for Sensor Networks (TPSN), which we explain in more detail in Section **Error! Reference source not found.**. Though many subtleties may exist in the implementation of this method, we present the scheme used by TPSN here.

In order to obtain a definitive relation between the two clocks with a single message exchange, two basic assumptions need to be made:

1) The offset between the clocks is constant in the small time period during the message exchange;
2) The propagation delay is the same in both directions.



*T*2, *T*3 are measured by node *B* clock

*T*1, *T*4 are measured by node *A* clock

Figure 9.1 Two-way message exchange between a pair of nodes.

Consider a two-way message exchange between nodes *A* and *B* as shown in **Error! Reference source not found.**. Node *A* initiates the synchronization by sending a packet at *T*1 (according to its local clock), which includes *A*'s current local time *T*1. *B* receives this packet (according to its local clock) at $T2 = T1 + \Delta + d$, where Δ is the relative clock offset between the two nodes, and d is the propagation delay of the pulse. *B* responds at time *T*3 with an acknowledgement packet, which includes the level number of *B* and the values *T*1, *T*2, and *T*3. Then, node *A* can calculate the clock offset and propagation delay as below and synchronize itself to *B*.

$$\Delta = \frac{(T2-T1)-(T4-T3)}{2},$$
$$d = \frac{(T2-T1)+(T4-T3)}{2}.$$

9.4.1.2 Reference Broadcast Synchronization

Reference Broadcast Synchronization (RBS) [3], proposed by Elson, Girod and Estrin, uses a *third party* for synchronization. Instead of synchronizing the sender with a receiver, this scheme synchronizes a set of receivers with one another. Although its application in sensor networks is novel, the idea of *receiver-receiver synchronization* was previously proposed for synchronization in broadcast environments [11]. In the RBS scheme, nodes send reference beacons to their neighbors. A reference beacon does not include a timestamp. Instead, its time of arrival is used by receiving nodes as a reference point for comparing clocks.

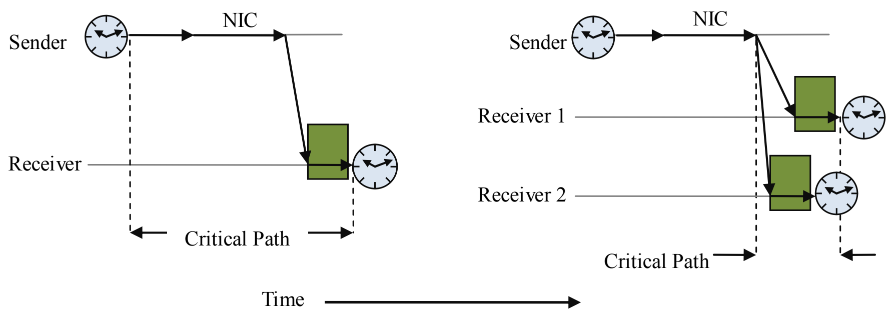


Figure 9.2 Comparison of traditional synchronization systems to RBS [3].

By removing the sender's non-determinism from the critical path (see **Error! Reference source not found.**), RBS may achieve better precision compared to traditional synchronization methods that use two-way message exchanges between synchronized nodes. As the sender's non-determinism has no effect on RBS precision, the only sources of errors can be the non-determinism in propagation time and receiving time. The authors claim that a single broadcast will propagate to all receivers at essentially the same time; hence the propagation error is negligible. This is especially true when the radio ranges are relatively small (compared to the speed of light timed by the required synchronization precision), as is the case for sensor networks [4]. Accordingly, they only account for the receiving time errors when analyzing the accuracy of their model.

In the simplest form of RBS, a node broadcasts a single pulse to two receivers. The receivers, upon receiving the pulse, exchange their receiving times of the pulse and try to estimate their relative offsets. This basic RBS scheme can be extended in two ways:

- Allowing synchronization between *n* receivers by a single pulse, where *n* may be larger than two;
- Increasing the number of reference pulses to achieve higher precision.

It is shown by simulations that 30 reference broadcasts (for a single synchronization in time) can improve the precision from 11 μs to 1.6 μs when synchronizing a pair of nodes. This redundancy can also be used for estimating clock skews. Instead of averaging the phase offsets from multiple observations (e.g., each of 30 reference pulses), one can perform a least squares linear regression to this data. Then the frequency and phase of the local node's clock with respect to the remote node can be recovered from the slope and intercept of the line, which is explained next for the Tiny-Sync protocol.

9.4.1.3 Tiny-Sync & Mini-Sync

Tiny-Sync and Mini-Sync are two lightweight synchronization algorithms proposed by Sichitiu and Veerarittiphan [2]. The authors assume that each clock can be approximated by an oscillator with a fixed frequency. As argued in Section 9.1.1, two clocks, $C_1(t)$ and $C_2(t)$, can be linearly related under this assumption as

$$C_1(t) = a_{12} \cdot C_2(t) + b_{12}, \tag{2}$$

where a_{12} is the relative drift, and b_{12} is the relative offset between the two clocks.

These algorithms use a method similar to the conventional two-way messaging scheme, but obtain a relation between clocks in a rather different way. Node 1 sends a probe message to node 2, timestamped with t_o, the local time just before the message is sent. Node 2 generates a timestamp when it gets the message at t_b, and immediately sends back a reply message[1]. Finally, node 1 generates a timestamp t_r as the time when it gets this reply message. Using the absolute order between these timestamps and Eq. (2), the following inequalities can be obtained:

$$\begin{aligned} t_o &< a_{12} \cdot t_b + b_{12}, & (3)\\ t_r &> a_{12} \cdot t_b + b_{12}. & (4) \end{aligned}$$

[1]Immediate reply assumption can be relaxed without loss of generality, but we skip that case here for brevity.

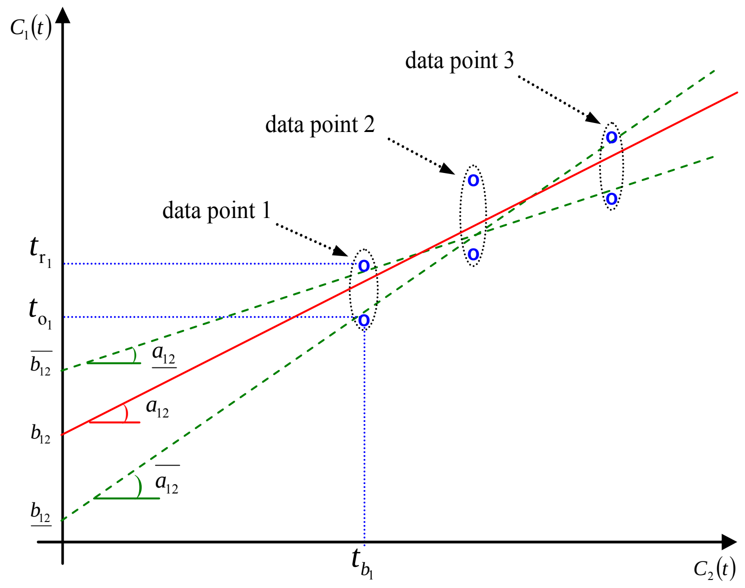


Figure 9.3 The constraints imposed on a_{12} and b_{12} by data points [2].

The 3-tuple of the timestamps (t_o, t_b, t_r) is called a *data point*. Tiny-Sync and Mini-Sync work with some set of data points, each collected by a two-way message exchange as explained. As the number of data points increases, so does the precision of the algorithms. Each data point corresponds to two constraints on the relative drift and relative offset (Eqs. (3) and (4)). The constraints imposed by data points are depicted in **Error! Reference source not found.**. Note that the line corresponding to Eq. (2) must lie between the vertical intervals created by each data point. One of the dashed lines in **Error! Reference source not found.** represents the steepest possible such line satisfying Eq. (2). This line gives the upper bound for relative drift, $\overline{a_{12}}$, (slope of the line) and the lower bound for relative offset, $\underline{b_{12}}$, (y-intercept of the line) between the two clocks. Similarly, the other dashed line gives the lower bound for relative drift ($\underline{a_{12}}$) and the upper bound for relative offset ($\overline{b_{12}}$). Then, the relative drift a_{12} and the relative offset b_{12} can be bounded as

$$\underline{a_{12}} \le a_{12} \le \overline{a_{12}},$$
$$\underline{b_{12}} \le b_{12} \le \overline{b_{12}}.$$

Exact drift and offset values cannot be determined by this method (or any other method as long as message delays are unknown), but can be well estimated by

$$a_{12} = \frac{\overline{a_{12}} + \underline{a_{12}}}{2} \pm \frac{\overline{a_{12}} - \underline{a_{12}}}{2},$$
$$b_{12} = \frac{\overline{b_{12}} + \underline{b_{12}}}{2} \pm \frac{\overline{b_{12}} - \underline{b_{12}}}{2}.$$

The tighter the bounds get, the higher the chance that the estimates will be good (i.e., the precision of synchronization gets higher as the above bounds get tighter). In order to tighten the bounds, one can solve the linear programming problem consisting of the constraints dictated by all data points, in order to get the optimal bounds resulting from the data points. However, the linear programming problem gets larger with the increasing number of data points and this approach is quite complex for sensor networks because it requires high computation and storage for keeping all data points in memory.

The basic intuition behind Tiny-Sync and Mini-Sync algorithms is the observation that not all data points are useful. Consider, for example, the three data points in **Error! Reference source not found.**; the intervals $[\underline{a_{12}}, \overline{a_{12}}]$ and $[\underline{b_{12}}, \overline{b_{12}}]$ are only bounded by data points 1 and 3. Therefore, data point 2 is useless in this example. Following this intuition, Tiny-Sync keeps only the four constraints - the ones that yield the best bounds on the estimates - among all data points. The resulting algorithm is much simpler than solving a linear programming problem. However, this scheme does not always give the optimal solution for the bounds. The algorithm may eliminate some data point, considering the data point useless, although it would actually give a better bound together with another data point that is yet to occur.

Mini-sync is an extension of Tiny-Sync that finds the optimal solution with an increase in complexity. The idea is to prevent the algorithm of Tiny-Sync from eliminating the constraints that might be used by some future data points to give tighter bounds. The authors argue by using experimental results that although sub-optimal, the performance of Tiny-Sync is comparable to that of the optimal Mini-sync.

9.4.2 Multi-hop Synchronization

A WSN typically spans a much larger area compared to the transmission range of the radio transmitters used; hence, data collected at a sink node may have originated at different nodes that are several hops apart. A common view of the time between such nodes can only be established through multi-hop synchronization protocols, which we review next.

9.4.2.1 Multi-hop RBS

We have presented in Section **Error! Reference source not found.** how RBS synchronizes a set of receivers in a single neighborhood. In many cases, the nodes that need synchronized time may not be in the coverage area of some common node. Then, some other nodes should act as gateways for time translation between neighborhoods to *route* the time information from one node to another.

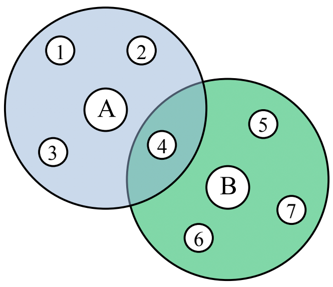


Figure 9.4 A topology where multi-hop time synchronization is required [3].

Figure 9.4 depicts a case where multi-hop synchronization is required. For example, node 1 and node 7 are not in the same neighborhood, i.e., they do not share a common sender from which they can both receive a synchronization pulse. In this case, node 4 acts as a gateway node between the two neighborhoods. When senders A and B broadcast synchronization pulses to their neighborhood as usual, node 4 gets both of these pulses and can thus relate the local clocks of A and B, i.e., the two neighborhoods. When a beacon sender broadcasts a synchronization pulse, it essentially creates a set of nodes (a neighborhood) in which nodes can relate their local clocks among each other. Now consider a graph whose vertices correspond to sensor nodes in the network. An edge between two vertices in this graph exists if the corresponding nodes in the network are within the same neighborhood formed by RBS, i.e., if the two nodes can receive synchronization pulses from the same beacon sender. Then multi-hop synchronization can be performed along the edges of this graph. To this end, the concept of "time routing in multi-hop networks" is introduced. Finding the shortest path between two nodes would yield a minimal error multi-hop synchronization path for this pair of nodes. Moreover, the authors proposed assigning weights to edges to represent the quality of pairwise synchronizations (e.g., using the residual error of the linear fit).

In the analysis of the multi-hop RBS algorithm, the authors argue that there is just a slow decay in precision by multi-hop synchronization; the average synchronization error is proportional to $\sqrt{n}$ for an n-hop network. Using the implementation of RBS on IPAQ and 802.11-based testbed, and including the kernel level timestamping, an error of 3.68 $\pm$ 2.57 μs was measured after 4 hops.

9.4.2.2 Timing-Sync Protocol for Sensor Networks (TPSN)

Ganeriwal *et al.* proposed a network-wide time synchronization protocol for sensor networks, called the Timing-Sync Protocol for Sensor Networks (TPSN) [5]. The protocol works in two phases: *level discovery* and *synchronization*. The aim of the first phase is to create a hierarchical topology in the network, where each node is assigned a level. Only one node is assigned level 0, the *root node*. In the second phase, a node of level *i* synchronizes to a node of level *i-1*. At the end of the synchronization phase, all nodes are synchronized to the root node, and the network-wide synchronization is achieved.

This protocol is like a practical adaptation of NTP [9], where each computer can simultaneously be a server for the computers lower in the hierarchy or a client of the computers higher in the hierarchy. The basic structural difference is that NTP makes use of the existing infrastructure in the Internet, while there is no infrastructure in a sensor network, and such a protocol needs to create a virtual hierarchy before applying the synchronization scheme.

- *Level Discovery Phase*: This phase is run once at the network deployment. First a node should be determined as the root node. This could be a sink node in the sensor network, and the sink may have a GPS receiver, in which case the algorithm will synchronize all nodes to an external time (time in the physical world). If such a sink is not available, sensor nodes can periodically take over the functionality of the root node. An existing leader election algorithm might be used for this periodic root node election step.

  The root node is assigned level 0, and initiates the level discovery phase by broadcasting a *level_discovery* packet. This packet contains the identity and level of the sender node. Upon receiving this packet, the neighbors of the root node assign themselves level 1. Then each level 1 node broadcasts a *level_discovery* packet with its level and identity in the packet. Once a node is assigned a level, it discards further incoming *level_discovery* packets. This broadcast chain goes on through the network, and the phase is completed when all nodes are assigned a level.

- *Synchronization Phase*: This phase is initiated by the root node's *time_sync* packet. On receiving this packet, level-1 nodes initiate a two-way message exchange with the root, as explained in Section 9.4.1.1.1.1. Before initiating the message exchange, each node waits for some random time in order to minimize collisions in the medium access. Once they get back a reply from the root node, they adjust their clocks to the root node. Level-2 nodes, overhearing some level-1 node's communication with the root, initiate a two-way message exchange with a level-1 node, again after waiting for some random time to ensure that level-1 nodes have completed their synchronization. This procedure eventually gets all nodes synchronized to the root node.

TPSN is implemented on Berkeley's Mica architecture [8] and makes use of timestamping packets at the MAC layer in order to reduce the uncertainty at the sender, as mentioned in Section 9.1.2. Ganeriwal *et al.* claim that TPSN achieves two times better precision than RBS and that the precision reported for RBS is due to using a superior operating system (Linux) and much more stable crystals available in IPAQs. Thus RBS is implemented on Mica sensor architecture as well as TPSN in order to compare their performances. They report an average of 29.13 μs precision for their implementation of RBS on Mica motes, while that of TPSN is 16.9 μs on the same hardware platform. Essentially, it is claimed that uncertainty at the sender contributes very little to the total synchronization error as it is minimized by the use of low-level timestamps at the sender and therefore the classical sender-receiver synchronization is more effective than receiver-receiver synchronization in sensor networks.

9.4.2.3 Lightweight Tree-based Synchronization (LTS)

Lightweight Tree-based Synchronization (LTS), proposed by Greunen and Rabaey [7], is distinguished from other work in the sense that the aim is not to maximize accuracy, but to minimize the complexity of the synchronization. Thus, the required synchronization accuracy is assumed to be given as a constraint and the target is to devise a synchronization algorithm with minimal complexity to achieve the given precision. This approach is supported by the claim of the authors that the maximum time accuracy required in sensor networks is relatively low (within fractions of a second). Therefore, it is sufficient to use a relaxed or lightweight synchronization scheme in sensor networks. Clearly, this assumption may not hold for some applications or services of sensor networks, such as measuring the time-of-flight of sound [12], forming a TDMA schedule [13], and distributing an acoustic beamforming array [14]**Error! Reference source not found.**, which require synchronized time with high precision. However, a loose synchronization may be acceptable in most other cases within the wide range of applications projected for WSNs.

Two LTS algorithms are proposed for multi-hop synchronization of the network based on the pair-wise synchronization scheme described in Section 9.4.1.1.1.1. Both algorithms require nodes to synchronize to some *reference node(s)*, such as a sink node in the sensor network. The first algorithm is centralized and needs a spanning tree to be constructed first. Then pair-wise synchronization is done along the $n-1$ edges of the spanning tree. In the centralized algorithm, the reference node is the root of the spanning tree and has the responsibility of initiating a *resynchronization* as needed. Using the assumption that the clock drifts are bounded and given the required precision, the reference node calculates the time period that a single synchronization step will be valid. Since the depth of the spanning tree affects the time to synchronize the whole network as well as the precision error at the leaf nodes, the depth of the tree is communicated back to the root node so that it can use this information in its resynchronization time decision.

The second multi-hop LTS algorithm performs network-wide synchronization in a distributed fashion. Each node decides the time for its own synchronization and a spanning tree structure is not used in this algorithm. When node i decides that it needs to synchronize (using the desired accuracy, its distance from the reference node, and the clock drift), it sends a synchronization request to the closest reference node (by any routing mechanism available). Then all nodes along the path from that reference node to i must be synchronized before node i can be synchronized. The advantage of this scheme is that some nodes may have less frequent events to deliver and therefore may not need frequent synchronization. Since nodes have the opportunity to decide on their own synchronization, this saves unnecessary synchronization effort for such nodes. On the other hand, letting each node decide on resynchronization may boost the number of pairwise synchronizations because for each synchronization request all nodes along the path from the reference node to the resynchronization initiator need to be synchronized. As the number of synchronization requests increase, the overall effect of synchronizations along these paths may be a significant waste of resources. Hence, the idea of aggregating synchronization requests is proposed; when any node wishes to request synchronization, it queries adjacent nodes to discover the existence of any pending request. If any exists, the synchronization request of this node could be aggregated to a pending request, decreasing the inefficiency that would be caused by two separate synchronizations along the same path.

The performance of LTS algorithms are tested by simulations of a connected ad-hoc network consisting of 500 nodes, placed uniformly at random in a $120\text{m} \times 120\text{m}$ rectangular area. The transmission range is set to 10m. It is assumed that there is a single reference node at the center of the area, which has access to an accurate time. All nodes should synchronize to this reference node. The required accuracy is determined as 0.5 seconds, and the simulation is executed for 10 hours. As a metric to evaluate the performance of the synchronization algorithms, the number of pair-wise synchronizations required to keep the network synchronized is analyzed. The average number of synchronizations required for each node is 36 for the centralized LTS over 10 hours of simulation time. If the number of participating nodes (the nodes that need synchronized time and thus participate in the algorithm) is low, the distributed algorithm performs much better; for 65% participation, the number of synchronizations per node drops to around 4-5 synchronizations when distributed LTS is used. Another metric is the average depth of the spanning tree and an average of 5 to 7 is reported for both algorithms.

9.4.2.4 Flooding Time Synchronization Protocol (FTSP)

The Flooding Time Synchronization Protocol (FTSP) [15] utilizes and enhances some key ideas from TPSN and RBS, and combines them with periodic flooding of synchronization messages to achieve network-wide synchronization, which is robust against node and link failures. FTSP implements MAC-layer timestamping as in TPSN and drift compensation with linear regression as in RBS.

A critical performance enhancement of FTSP over prior work is due to its focus on the detailed analysis of the transceiver pipeline in the wireless channel. FTSP uses a single message per synchronization and introduces the use of multiple timestamps for a single synchronization message. Timestamps are made at each byte boundary as they are transmitted or received and the final timestamp on the message is computed by an average of the normalized timestamps. This effectively reduces the jitter of the interrupt handling and encoding/decoding times through the CPU, radio and antenna of the sender and the receiver. Though the achievable error correction using this technique is bounded by the number of bytes, an experiment on the Mica2 platform reports roughly a 10-fold improvement in the precision with only six timestamps.

In FTSP, all nodes in the network synchronize to a dynamically (re)elected *root* node by the use of controlled flooding. Similar to the *data points* in Tiny-Sync described in Section **Error! Reference source not found.**, nodes use *reference points* for synchronization, each of which is a pair of local and "global" timestamps. A reference point is collected by receiving a synchronization message from the *root* or another node that is previously synchronized to the root. When a node gathers enough reference points, it performs synchronization by estimating its clock drift and offset using linear regression, after which it can also start broadcasting synchronization messages.

A synchronization message contains three fields: *timeStamp*, *rootID*, and *seqNum*. The *timeStamp* is the synchronized "global" time as estimated by the sender of this synchronization message. The *rootID* field contains the unique ID of the root node as currently known to the sender of this message. The *seqNum* is used to control the flooding of messages and is incremented at every synchronization round initiated by the root node. A node uses only the first message arrived for each *rootID* and *seqNo* pair. When a node does not receive synchronization messages for certain duration of time, it declares itself as the root. In order to eliminate the problem of having multiple roots, a node that receives a message with smaller *rootID* gives up its root status; hence, only the node with the smallest ID remains as the single root.

The experiments with an FTSP implementation on Mica2 motes report an average synchronization error of 3μs in a 6-hop network, resulting in a 0.5μs per hop accuracy, which is evidently better than that of RBS and TPSN. FTSP is also reported to use less network resources than the other two protocols; if the resynchronization period is T seconds, then each node sends 1 message per T seconds in FTSP, 2 messages per T seconds in TPSN (1 message to parent and 1 response) and 1.5 message per T seconds in RBS (0.5 for a reference broadcast and 1 for a time stamp exchange message).

9.4.3 Long-term Synchronization

The synchronization protocols presented so far mainly aim to provide a common timescale between clocks at a given instant. However, as argued earlier, the achieved harmony of the clocks may be quickly disrupted by varying clock drifts. The most straightforward method for achieving time synchronization over long durations (such as the lifetime of a sensor network) is the periodic application of one of those schemes that provide instantaneous synchronization. As a better alternative, adaptive schemes carefully designed for long-term synchronization have been proposed for better use of limited resources and/or higher precision in WSNs.

9.4.3.1 Post-facto Synchronization

Post-facto synchronization was a pioneering work by Elson and Estrin [16-17], which has led afterwards to their RBS scheme. They proposed that unlike in traditional synchronization schemes such as NTP [9], local clocks of the sensor nodes should normally run unsynchronized in their own pace and should synchronize only when necessary. This way, local timestamps of two nodes at the occurrence time of an event are synchronized later by extrapolating backwards to estimate the offset between clocks at a previous time (at the time of the event). Post-facto synchronization can also be termed as *reactive synchronization*, while the traditional schemes are *proactive*, requiring the clocks of sensor nodes to be synchronized before an event of interest occurs.

9.4.3.2 Time-Diffusion Synchronization Protocol (TDP)

The Time-Diffusion Protocol [18] is a network-wide synchronization protocol which maintains an equilibrium time throughout the network, allowing only a small deviation from the equilibrium. The deviation tolerance can be adjusted based on the specific sensor network application.

TDP achieves long-term synchronization by defining *active* and *inactive* periods (see

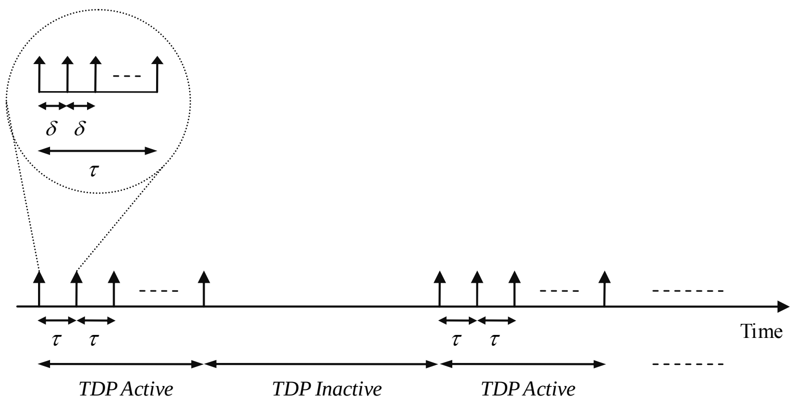


Figure **9.5**). At every τ seconds during the active period, some nodes are elected as *master* nodes that broadcast timing information to their neighbors at every δ seconds. Nodes receiving timing information from the master nodes self-determine to become *diffused leader* nodes that further broadcast the timing information to their neighbors. Neighbors of diffused leaders may also become diffused leader nodes, spreading the timing information further away from the current master nodes. Therefore, in effect tree-like structures are temporarily created for diffusing the time information from the master nodes to the rest of the network.

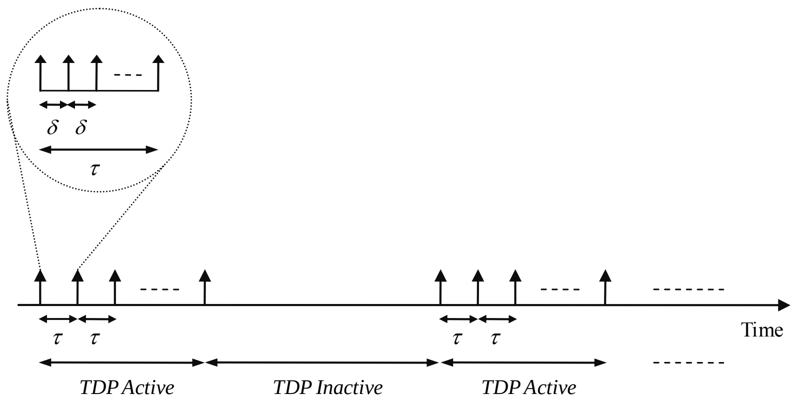


Figure 9.5 Duty cycle of TDP with its active/inactive schedule [18].

The distributed autonomous diffusion process combined with the periodic reelection of master nodes provide network-wide synchronization with tunable parameters τ and δ, which together determine the length of the TDP active period. The appropriate duration of the active period depends on the desired synchronization accuracy throughout the network, whereas the duration of the inactive period is dictated by how much the clocks are allowed to drift from each other in the worst case.

9.4.3.3 Rate Adaptive Time Synchronization (RATS)

Rate Adaptive Time Synchronization (RATS) by Ganeriwal *et al.* [19] is an energy-efficient long-term synchronization protocol that can adapt to variable clock drifts while achieving the precision specified by the application. The design of RATS is based on an in-depth analysis of empirical measurements to investigate the interplay between three key parameters affecting the long-term synchronization, including synchronization rate, history of synchronization data, and the estimation scheme.

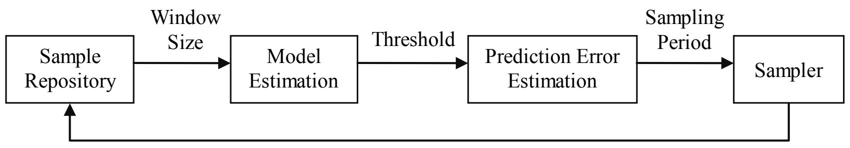


Figure 9.6 Time synchronization control loop for RATS.

The objective of the RATS protocol is to dynamically implement the control loop illustrated in Figure 9.6. The samples repository in the figure represents the synchronization data points (“observations”) collected by a node in relation to another node's local clock. A window of these samples is input to an estimator that uses the observations to estimate the relative model between the two clocks. A new sampling period is then calculated based on the comparison of the prediction error of this model to the application-specific error bound. A detailed analysis of the long-term empirical measurements guides the choice or learning of RATS parameters such as the optimal window size.

The experiments with a RATS implementation on Mica2 motes show that for an error bound of 225 microseconds the average sampling period is roughly 30 minutes. The main use proposed for the RATS protocol is more efficient duty-cycling in sensor networks. Hence, it is integrated with a MAC layer protocol, BMAC [20], which does not assume any time synchronization but instead uses a very long preamble to guarantee that the receiver wakes up before the actual data transmission. The integration of RATS in BMAC enables the use of shorter preambles and longer duty cycles, which is reported to provide an order of magnitude reduction in energy consumption of a node with a negligible impact on the packet loss rate.

9.4.4 Other Protocols and Relevant Work

Younis and Fahmy proposed a pair of distributed protocols, SYNC-IN & SYNC-NET [21], for synchronization of clustered sensor networks. The goal is to provide an end-to-end synchronization between communicating nodes, rather than a global timescale throughout the network. It is assumed that the network is clustered using any clustering approach and that nodes can tune their transmission power. SYNC-IN protocol is used for intra-cluster synchronization using the smaller in-cluster transmission power, where the nodes are synchronized to the cluster heads. The cluster heads are synchronized with the SYNC-NET protocol using a higher transmission power.

The asynchronous diffusion protocol proposed by Li and Rus [22] provides a simple approach for synchronization. Though diffusion-based as in TDP, it is completely asynchronous; nodes average time readings obtained from their neighbors and then broadcast the computed value as their updated clock reading. One drawback of this method is that a clock may run backwards after an update, causing the same time reading to occur more than once.

In [6], a message ordering scheme for sensor networks is proposed. The intention is not to synchronize clocks but to reason about the relative order between messages or events. The scheme described in this work complies with the most relaxed version of synchronization and is not applicable to most synchronization needs in WSNs.

In a study that is more of theoretical interest [23], the authors considered an infinitely large sensor network and proposed an approach in which nodes collaborate to generate a waveform that carries enough synchronization information to all nodes in the network. They argued that as the number of nodes goes to infinity, optimal synchronization is possible at a reasonable complexity.

A CENS[2] technical report also presents a study on optimal and global time synchronization in sensor networks [4]. They considered the problem of finding the best path (chain) of pair-wise synchronizations that would yield the optimum synchronization between any pair of nodes in the network. They claimed that an appropriately-weighted combination of alternating paths for synchronization should yield better precision than any single path could provide. By the use of such weighted combination of paths, the optimal global synchronization problem can be abstracted as a network flow formulation. In this work, the authors did not aim at giving a practical synchronization method, but presented the theoretical results for optimal global synchronization, which can be used as a reference to compare the performance of global synchronization methods.

Reachback Firefly Algorithm (RFA) [24] is a distributed synchronicity algorithm inspired by the Mirollo-Strogatz (MS) mathematical model [25], which was previously proposed for explaining how neurons and fireflies spontaneously synchronize. The main goal of RFA is not time synchronization but synchronicity (defined as the ability for all nodes in the network to agree on a common period and phase for firing pulses). However, synchronicity can be used as a primitive to obtain time synchronization. Though it presents a new approach for time synchronization, RFA has significant overhead and its performance is not yet evaluated as a time synchronization protocol.

Another biologically-inspired algorithm DESYNC [26] unconventionally uses the MS model for *desynchronization*, a primitive introduced by the authors as the logical opposite of synchronization. Instead of performing periodic tasks at the same time, the nodes try to schedule tasks so that they are as far away from each other as possible. In other words, the firing events are evenly distributed in a given time frame, rather than having them coincide at the same instant. This primitive can be used for many sensor networking tasks, such as periodic resource sharing, distributed collaborative sensing, and channel access scheduling.

Adaptive-Rate Synchronization Protocol (ARSP) [27] addresses not only adjusting the local clock values at nodes, but also the synchronization intervals to ensure that the synchronization errors remain within a given tolerance with high probability. Motivated by the energy limitations and various precision needs of sensor network applications, ARSP aims to provide a tunable tool for different scenarios. It utilizes both two-way message exchange and receiver-receiver synchronization primitives, while the synchronization intervals are adjusted at run time as a function of the in-tolerance probability of the various nodes.

---

[2]CENS: Center for Embedded Networked Sensing - University of California, Los Angeles

Reference [28] gives an overview of the time synchronization problem in WSNs, and defines the requirements and various issues for designing a synchronization algorithm for WSNs. The authors argued that such an algorithm should be multi-modal, tiered and tunable so that it can satisfy the diverse needs of various sensor network applications. Moreover, they suggest that the local clock of each node be free-running, i.e., one should not adjust the local clock. Instead, the synchronization scheme should build up a table of parameters that enables each node to convert its local clock to that of another, and vice versa.

9.5 Summary and Future Directions

In this chapter, we have introduced the synchronization problem and common challenges for synchronization, discussed the need for synchronization and requirements of synchronization methods in WSNs, and reviewed the major synchronization methods and protocols for WSNs. The two synchronization protocols, RBS and TPSN, both report very high precisions on the orders of a few microseconds although they use completely different approaches. The receiver-receiver synchronization of RBS completely eliminates the uncertainty at the sender by using a third party node, while TPSN minimizes this uncertainty by low-level timestamping at the sender. On the other hand, the receiver-receiver synchronization requires four messages sent and three messages received for synchronizing two nodes, while the sender-receiver synchronization requires only two sent and two received messages. As radio communication is known to be the most energy consuming component of sensor node operations, this is almost a two times increase in energy complexity. This increase in the complexity of receiver-receiver synchronization can be reduced to some degree by synchronizing many receivers by a single synchronization pulse broadcast by the sender. Although TPSN does not suffer from energy complexity in this respect, it needs a hierarchical structure of nodes to be formed, which might increase the synchronization cost. FTSP combines and enhances the key ideas of TPSN and RBS and has superior performance compared to both, with smaller communication overhead. LTS algorithms offer very low-cost synchronization, however, with very limited accuracy and thus limited applicability.

Studying the long-term behavior of synchronization schemes for WSNs is a rather less explored area. Most protocols suggest periodic reapplication of instantaneous synchronization methods, which can be costly for resource-constrained sensor networks. We have presented three different approaches for efficiently maintaining synchronized time in the network for longer time periods. Before concluding the chapter, we review some open issues and possible research directions in this field.

Most of the time synchronization work in the literature analyzes and presents their results based on experiments or simulations. For single-hop synchronization, there are also *analytical models* to define the accuracy characteristics of a proposed synchronization scheme. However, there is a lack of analytical models for multi-hop synchronization. When two nodes apart are synchronized using multiple pairwise synchronization steps, errors are usually expected to grow. However, since the pairwise errors may have different signs and magnitudes, the overall effect of multi-hop synchronization is usually much smaller than the sum of magnitudes of single-hop errors. An analytical model for this artifact may be developed, accounting for the probabilistic variations in the sign and magnitude of single-hop synchronization errors.

Identification or discovery of nodes to act as beacon senders in RBS is an important issue. If there is more than one beacon sender in a single neighborhood, the resulting redundancy may be used to not only improve precision but also increase the consumption of limited resources in the network. The correlation between this redundancy and precision may be investigated, and methods for identifying beacon senders to achieve some desired point in this trade-off curve proposed.

Extensive research on sensor networks boosts the evolution of these systems. Although sensor networks are mostly considered as fixed topologies (with stationary sensor nodes), and sensor network protocols so far usually assume that the nodes are stationary, next generation sensor networks may be expected to include mobile sensor nodes. Indeed, the Networked Infomechanical Systems (NIMS) project is a recent initiative towards this, and has already announced the development and deployment of initial prototypes that operated in a forest field biology station[3]. As such systems evolve; synchronization methods that take mobility into account will be needed. Global synchronization protocols may even benefit from the mobility, as mobile nodes will "carry" time information from one part of the network to other parts, potentially increasing global synchronization accuracy.

References

[1] K. Römer, "Time synchronization in ad hoc networks," in *Proc. of 2001 ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'01)*, Long Beach, CA, Oct. 2001, pp. 173-182.
[2] M. L. Sichitiu and C. Veerarittiphan, "Simple, accurate time synchronization for wireless sensor networks," in *Proc. of 2003 IEEE Wireless Communications and Networking Conference (WCNC'03)*, New Orleans, LA, Mar. 2003, pp. 1266-1273.
[3] J. Elson, L. Girod, and D. Estrin, "Fine-grained time synchronization using reference broadcasts," in *Proc. of the 5th Symposium on Operating Systems Design and Implementation (OSDI'02)*, Boston, MA, Dec. 2002, pp. 147-163.
[4] R. Karp, J. Elson, D. Estrin, and S. Shenker, "Optimal and global time synchronization in sensornets," Technical Report 0009, Center for Embedded Networked Sensing, University of California, Los Angeles, CA, Apr. 2003.
[5] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing sync protocol for sensor networks," in *Proc. of ACM SenSys'09*, Los Angeles, CA, Nov. 2003, pp. 138-149.
[6] K. Römer, "Temporal message ordering in wireless sensor networks," in *Proc. of IFIP MedHocNet'03*, Mahdia, Tunisia, Jun. 2003, pp. 131-142.
[7] J. V. Greunen and J. Rabaey, "Lightweight time synchronization for sensor networks," in *Proc. of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA'03)*, San Diego, CA, Sept. 2003, pp. 11-19.
[8] J. Hill and D. Culler, "A wireless embedded sensor architecture for system-level optimization," Technical Report, U.C. Berkeley, May 2001.
[9] D. Mills, "Network time protocol (version 3) specification, implementation and analysis," RFC 1305, available at http://www.faqs.org/ftp/rfc/rfc1305.pdf, Mar. 1992.
[10] E. D. Kaplan, "Understanding GPS: Principles and Applications," Artech House, Norwood, MA, Feb. 1996.
[11] P. Verissimo and L. Rodrigues, "A posteriori agreement for fault-tolerant clock synchronization on broadcast networks," in *Proc. of the 22th International Symposium on Fault-Tolerant Computing*, Boston, Jul. 1992, pp. 527-536.
[12] L. Girod and D. Estrin, "Robust range estimation using acoustic and multimodal sensing," in *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'01)*, Maui, HI, Mar. 2001, pp. 1312-1320.
[13] G. Asada, M. Dong, T. S. Lin, F. Newberg, G. Pottie, W. J. Kaiser, and H. O. Marcy, "Wireless integrated network sensors: low power systems on a chip," in *Proc. of the European Solid State Circuits Conference*, Den Hague, The Netherlands, Sept. 1998, pp. 9-16.
[14] H. Wang, L. Yip, D. Maniezzo, J. C. Chen, R. E. Hudson, J. Elson, and K. Yao, "A wireless time-synchronized COTS sensor platform part II-applications to beamforming," in *Proc. of IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, CA, Sept. 2002.

[3]More information is available online at http://research.cens.ucla.edu/research/

[15] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The flooding time synchronization protocol," in *Proc. of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, Nov. 2004, pp. 39-49.

[16] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *Proc. of IEEE IPDPS Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, San Francisco, CA, Apr. 2001, pp. 1965-1970.

[17] J. Elson, "Time synchronization in sensor networks," PhD Thesis, University of California Los Angeles, 2003.

[18] W. Su and I. F. Akyildiz, "Time-diffusion synchronization protocol for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 13, no. 2, Apr. 2005, pp. 384-398.

[19] S. Ganeriwal, D. Ganesan, H. Shim, V. Tsiatsis, and M. B. Srivastava, "Estimating clock uncertainty for efficient duty-cycling in sensor networks," in *Proc. of the 3rd ACM SenSys Conference*, San Diego, CA, Nov. 2005, pp. 130-141.

[20] J. Polastre, J. Hill, and D. Culler, "Versatile low power medium access for wireless sensor networks," in *Proc. of the ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, Baltimore, MD, Nov. 2004, pp. 95-107.

[21] O. Younis and S. Fahmy, "A scalable framework for distributed time synchronization in multi-hop sensor networks," in *Proc. of 2005 IEEE Sensor and Ad Hoc Communications and Networks (SECON'05)*, Santa Clara, CA, Sept. 2005, pp. 13-23.

[22] Q. Li and D. Rus, "Global clock synchronization in sensor networks," *IEEE Transactions on Computers*, vol. 55, no. 2, Feb. 2006, pp. 214-226.

[23] A. Hu and S. D. Servetto, "Asymptotically optimal time synchronization in dense sensor networks", in *Proc. of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA'03)*, San Diego, CA, Sept. 2003, pp. 1-10.

[24] R. Nagpal, A. Patel, G. Tewari, M. Welsh, and G. Werner-Allen, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *Proc. of the 3rd International Conference on Embedded Networked Sensor Systems*, San Diego, CA, Nov. 2005, pp. 142-153.

[25] R. Mirollo and S. Strogatz, "Synchronization of pulse-coupled biological oscillators," *SIAM*, vol. 50, no. 6, Dec. 1990, pp. 1645-1662.

[26] J. Degesys, I. Rose, A. Patel, and R. Nagpal, "DESYNC: Self-organizing desynchronization and TDMA on wireless sensor networks," in *Proc. of the 6th International Conference on Information Processing in Sensor Networks (IPSN'07)*, Cambridge, MA, Apr. 2007, pp. 11-20.

[27] D. Macii and D. Petri, "An adaptive-rate time synchronization protocol for wireless sensor networks," in *Proc. of the IEEE Instrumentation and Measurement Technology Conference (IMTC'07)*, Warsaw, Poland, May 2007, pp. 1-6.

[28] J. Elson, and K. Römer, "Wireless sensor networks: A new regime for time synchronization," in *Proc. of the 1st Workshop on Hot Topics in Networks (HotNets-I)*, Princeton, NJ, Oct. 2002, pp. 149-154.