Time Synchronization in Sensor Networks: A Survey

Fikret Sivrikaya
Department of Computer Science
Rensselaer Polytechnic Institute, NY
sivrif@rpi.edu

Bülent Yener
Department of Computer Science
Rensselaer Polytechnic Institute, NY
yener@cs.rpi.edu

Abstract

Time synchronization is an important issue in multihop, ad-hoc wireless networks such as sensor networks. Many applications of sensor networks need local clocks of sensor nodes to be synchronized, requiring various degrees of precision. Some intrinsic properties of sensor networks such as limited resources of energy, storage, computation, and bandwidth, combined with potentially high density of nodes make traditional synchronization methods unsuitable for these networks. Hence there has been an increasing research focus on designing synchronization algorithms specifically for sensor networks. This paper reviews the time synchronization problem and the need for synchronization in sensor networks, then presents in detail the basic synchronization methods explicitly designed and proposed for sensor networks.

1 Introduction

As the advances in technology have enabled the development of tiny, low power devices capable of performing sensing and communication tasks, *sensor networks* emerged and received high attention of many researchers. Sensor networks are a special type of ad-hoc networks, where wireless devices (usually referred as nodes in the network) get together and spontaneously form a network without the need for any infrastructure. Because of the lack of infrastructure, such as routers in traditional networks, nodes in an ad-hoc network cooperate for communication, by forwarding each other's packets for delivery from a source to its destination. This yields a multihop communication environment. Though being a special type of ad-hoc networks, sensor networks have their own characteristics, such as much limited energy sources, high density of node deployment, cheap and unreliable sensor nodes. Having these extra limiting factors for their operation, sensor networks are designed to perform complex tasks such as emergency applications, environment monitoring, information gathering in battle fields, and many other uses, connecting the physical world to the virtual world of computers.

As in all distributed systems, time synchronization is an important component of a sensor network. Time synchronization in a computer network aims at providing a common time scale for local clocks of nodes in the network. Since all hardware clocks are imperfect, local clocks of nodes may drift away from each other in time, hence 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. This work presents a survey of existing research on time synchronization in the field of wireless sensor networks. To the best of our knowledge, there is no previous work that solely surveys this specific area.

The rest of this paper is organized as follows: In section 2 we describe how computer clocks operate and review the synchronization problem on computer networks. We also present common sources of error/inaccuracy in synchronization systems in this section. Section 3 presents motivations for studying time synchronization in sensor networks. Section 4 is devoted to detailed analysis of synchronization methods proposed for sensor networks so far. Last section concludes the paper with some remarks and future research directions.

2 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. The angular frequency of the hardware oscillator determines the rate at which the clock runs. The rate of a perfect clock, which can be denoted as $\frac{dC}{dt}$, would equal 1, however all clocks are subject to a clock drift; 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 fixed frequency [8]. Then, for some node i in the network, we can approximate its local clock as:

$$C_i(t) = a_i t + b_i, \tag{1}$$

where $a_i(t)$ is the clock *drift*, and $b_i(t)$ 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 equation (1), we can compare the local clocks of two nodes in a network, say node 1 and node 2 as:

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

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, then their relative drift is 1 -meaning the clocks have the same rate- and their relative offset is zero -meaning 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 (e.g. [5]). Also, the "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 $i = 1..n$, or it can be *local*; trying to equalize $C_i(t)$ for some set of nodes -mostly the ones that are spatially close. Equalizing just the instantaneous values (correcting the offsets) of clocks is not enough for synchronization since the clocks will drift away afterwards. Therefore a synchronization scheme should either equalize the clock rates as well as offsets, or it should repeatedly correct the offsets to keep the clocks synchronized over a time period.

The above definition of synchronization actually defines the most strict 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 needs of an application. In general, the synchronization problem can be classified into three basic types [9]. First -and the simplest- form of synchronization deals only with ordering of events or messages. The aim of such an algorithm is to be able to tell whether an event E_1 has occurred before or after another event E_2, i.e. just to compare the local clocks for order rather than having them synchronized. The algorithm proposed in [10] is an example to this type of synchronization. Second type of synchronization algorithms target maintaining relative clocks. In this scheme, nodes run their local clocks independently, but they keep information about the relative drift and offset of their 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 schemes proposed for sensor networks use this model [5, 6, 8]. The third -and most complex- form 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 algorithms is to preserve a global timescale throughout the network. The synchronization scheme of [9] conforms to this model, but the use of "always on" mode is not mandatory in the scheme.

2.1 Common Challenges for Synchronization Methods

All network time synchronization methods rely on some sort of message exchange between nodes. Nondeterminism 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 amount of 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 node. We can basically decompose the sources of error 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 (such as context switches), and the time to transfer the message to the network interface for transmission.

- **Access Time:** Each packet faces some delay at the MAC (Medium Access Control) 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 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.

3 The Need For Synchronization in Sensor Networks

There are several reasons for addressing the synchronization problem in sensor networks. 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 that they sense the vehicle to a sink node which in turn combines these information to estimate the location and velocity of the vehicle. Clearly, if the sensor nodes lack a common timescale (i.e., they 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 power-saving mode by turning off their sensors and/or transceivers) at appropriate times, and wake up when necessary. When using power-saving modes, 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 a 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 NTP or GPS are not suitable for use in sensor networks because of complexity and energy issues, cost and size factors. NTP works well 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 attach on cheap sensor devices, and GPS service may not be available everywhere, such as inside the buildings or under the water. Furthermore in adversarial environments, the GPS signals may not be trusted.

3.1 Requirements on the Synchronization Schemes for 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 on sensor networks. However, there are trade offs between the requirements of an efficient synchronization solution (e.g., *precision* versus *energy efficiency*), thus a single scheme may not satisfy them altogether.

- *Energy Efficiency* — As with all of the protocols designed for sensor networks, synchronization schemes should take into account the limited energy resources contained 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 the 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 μsecs.

- *Robustness* — A sensor network is typically left unattended for long times 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.

- *Scope* — The synchronization scheme may provide a global time-base for all nodes in the network, or provide local synchronization only among spatially close nodes. Because of the 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 for aggregating 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 a relatively large or expensive hardware (such as a GPS receiver) on a small, cheap device is not a logical option for synchronizing sensor nodes. The synchronization method for sensor networks should be developed with limited cost and size issues 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 this kind of applications, the network can not tolerate any kind of delay when such an emergency situation is detected. This is called the immediacy requirement, and might prevent the protocol designer from relying on excessive processing after such an event of interest occurs, which in turn requires that nodes be *pre-synchronized* at all times.

4   Synchronization Methods for Sensor Networks

Time synchronization in sensor networks has attracted attention in last few years. *Post-facto synchronization* was a pioneering work by Elson and Estrin [2]. They proposed that unlike in traditional synchronization schemes such as NTP, local clocks of the sensor nodes should normally run unsynchronized – in their own pace, but should synchronize whenever 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). This synchronization scheme has led afterwards to their RBS (Reference Broadcast Synchronization) protocol, which is discussed in detail in section 4.1.

Römer presented a synchronization algorithm for ad hoc networks [3] around the same time as [2], suggesting the timestamps in the messages to be transformed between nodes, instead of adjusting the clocks. However, his scheme used the traditional two-way message exchange for drift and offset estimation, with the assumption that the drifts are bounded by some constant ρ. This synchronization scheme achieves around 1ms precision with little overhead.

In [1], the authors report 2μs precision for synchronization, achieved by tight coupling between application and MAC layers in the protocol stack. The achievement in precision is basically due to the proposed architecture that enables timestamping messages at the instant the message is actually sent at the MAC layer, thereby eliminating uncertainties due to the sender.

[4] gives an overview of the time synchronization problem in sensor networks, defining the requirements, and various issues for designing synchronization algorithms for sensor networks. The authors argue 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 should be free-running, i.e. one should not adjust the local clocks. Instead, the synchronization scheme should build up a table of parameters that enables each node to convert its local clock to that of another node, and vice versa.

In [10] a message ordering scheme for sensor networks is proposed. The intention is not to synchronize clocks but to be able 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 for most synchronization needs in sensor networks.

A recent interesting study has a more theoretical approach to the problem [7]. In this work, the authors consider an infinitely large sensor network, and propose an approach in which nodes collaborate to generate a waveform that carries enough synchronization information to all nodes in the network. They argue that as the number of nodes goes to infinity, optimal synchronization is possible at reasonable complexity.

In the rest of this section, we present in detail the synchronization methods explicitly designed and proposed for sensor networks.

4.1 Reference Broadcast Synchronization (RBS)

Elson, Girod and Estrin [5] proposed a synchronization scheme for sensor networks, Reference Broadcast Synchronization (RBS), where their simple yet novel idea is to use a "third party" for synchronization: instead of synchronizing the sender with a receiver (as in most of the previous work), their 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). In RBS scheme, nodes send reference beacons to their neighbors. A reference beacon does not include a timestamp, but instead, its time of arrival is used by receiving nodes as a reference point for comparing clocks.
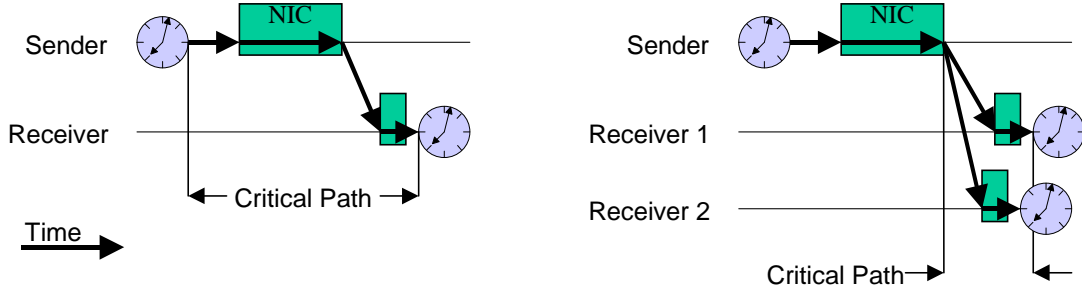

Figure 1: Comparison of traditional synchronization systems to RBS. [5]

The authors argue that, by removing the sender's nondeterminism from the critical path (figure 1), RBS achieves much better precision compared to traditional synchronization methods that use two-way message exchanges between synchronizing nodes. As the sender's nondeterminism has no effect on RBS precision, the only sources of error can be the nondeterminism in propagation time and receive time. The authors claim that a single broadcast will propagate to all receivers at essentially the same time, and hence the propagation error is negligible. This is especially true when the radio ranges are relatively small (compared to speed of light times the required synchronization precision), as is the case for sensor networks. So they only account for the receive time errors when analyzing 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 phase offsets. This basic RBS scheme can be extended in two ways: 1) allowing synchronization between n receivers by a single pulse, where n may be larger than two, 2) increasing the number of reference pulses to achieve higher precision. The authors show by simulation that 30 reference broadcasts (for a single synchronization in time) can improve the precision from $11\mu s$ to $1.6\mu s$ when synchronizing a pair of nodes. They also make use of this redundancy for estimating clock skews: instead of averaging the phase offsets from multiple observations (e.g. each of 30 reference pulses), they propose to 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.

5

The authors implemented and tested RBS on two different hardware platforms to acquire its precision performance. One platform is the Berkeley Motes, one of the widely used sensor node architectures. On Berkeley Motes, the achieved precision is within $11\mu s$. The other platform is commodity hardware, Compaq IPAQs running Linux kernel v2.4, connected with an 11 Mbit/sec 802.11 network. The reported precision achieved by RBS on this platform is $6.29 \pm 6.45\mu s$.

4.2 Timing-Sync Protocol for Sensor Networks (TPSN)

Ganeriwal et.al. proposed a network-wide time synchronization protocol for sensor networks, which they call Timing-Sync Protocol for Sensor Networks (TPSN) [9]. Their protocol works in two phases: "level discovery phase" and "synchronization phase". 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, called 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.

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 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.
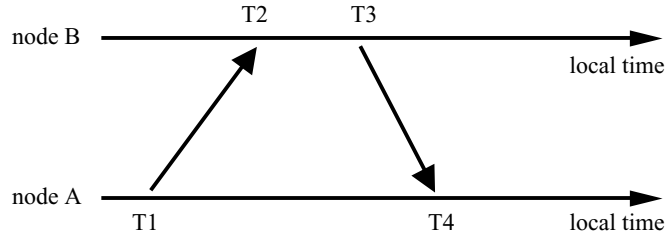

Figure 2: Two way message exchange between a pair of nodes.

Synchronization Phase — The basic building block of the synchronization phase is the two-way message exchange between a pair of nodes. The authors assume that the clock drift between a pair of nodes is constant in the small time period during a single message exchange. The propagation delay is also assumed to be constant in both directions. Consider a two-way message exchange between nodes A and B as shown in figure 2. Node A initiates the synchronization by sending a *synchronization_pulse* packet at $T1$ (according to its local clock). This packet includes A's level number, and the value $T1$. B receives this packet (according to its local clock) at $T2 = T1 + \Delta + d$, where Δ is the relative clock drift between the nodes, and d is the propagation delay of the pulse. B responds at time $T3$ with an acknowledgement packet, which includes the level number of B and the values $T1$, $T2$, and $T3$. Then, node A can calculate the clock drift and propagation delay as below, and synchronize itself to B.

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

The synchronization 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. Before initiating the message

exchange, each node waits for some random time, in order to minimize collisions on the wireless channel. 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 [1], and makes use of timestamping packets at the MAC layer in order to reduce *uncertainty at sender*. Ganeriwal et.al. claim that TPSN achieves two times better precision than RBS. They state that the precision of $6.5\mu s$ 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 performance. As we presented in section 4.1, RBS has actually been tested on Berkeley motes (by Elson et.al.), and the reported precision was $11\mu s$. However, Ganeriwal et.al. report, on average, $29.13\mu s$ precision for their implementation of RBS on Mica. The average error of TPSN is $16.9\mu s$ with its implementation 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.

4.3 Tiny-Sync and Mini-Sync

Tiny-Sync and Mini-Sync are the two lightweight synchronization algorithms, proposed mainly for sensor networks, by Sichitiu and Veerarittiphan [8]. The authors assume that each clock can be approximated by an oscillator with fixed frequency. As argued in section 2, two clocks, $C_1(t)$ and $C_2(t)$, can be linearly related under this assumption as:

$$C_1(t) = a_{12} \cdot C_2(t) + b_{12}, \tag{3}$$

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

Both algorithms use the conventional two-way messaging scheme to estimate the relative drift and offset between the clocks of two nodes; 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. Finally, node 1 generates a timestamp t_r when it gets this reply message. Using the absolute order between these timestamps and equation (3), the following inequalities can be obtained:

$$t_o \quad < \quad a_{12} \cdot t_b + b_{12} \tag{4}$$
$$t_r \quad > \quad a_{12} \cdot t_b + b_{12} \tag{5}$$

The 3-tuple of the timestamps (t_o, t_b, t_r) is called a "data point". Tiny-sync and mini-sync works with some set of data points, each collected by a two-way message exchange as explained. As the number of data points increases, the precision of the algorithms increases. Each data point corresponds to two constraints on the relative drift and relative offset (equations 4, 5). The constraints imposed by data points are depicted in figure 3. Note that the line corresponding to equation (3) must lie between the vertical intervals created by each data point. One of the dashed lines in figure 3 represent the steepest possible such line, satisfying equation (3). This line gives the upper bound for the relative drift (slope of the line, $\overline{a_{12}}$), and the lower bound for the relative offset (y-intercept of the line, $\underline{b_{12}}$) 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:

$$\underline{a_{12}} \quad \leq a_{12} \leq \quad \overline{a_{12}}$$
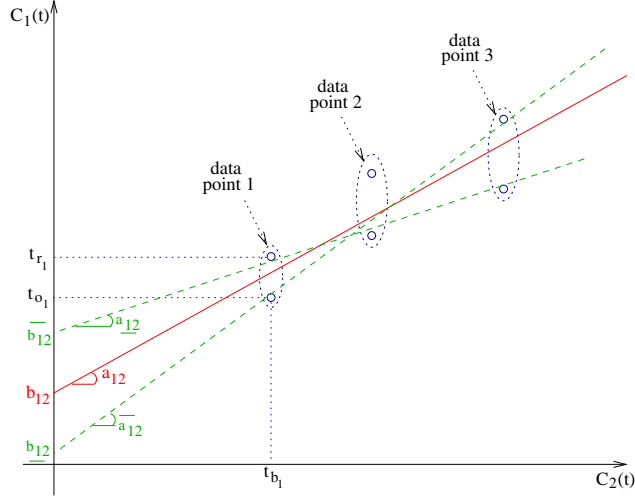$$\underline{b_{12}} \quad \leq b_{12} \leq \quad \overline{b_{12}}.$$

7

Figure 3: The constraints imposed on a_{12} and b_{12} by data points. [8]

The authors argue that exact drift and offset values can not be determined by this method (or any other method - as long as message delays are unknown), but they can be well estimated.

The tighter the bounds get, the higher the chance that the estimates will be good, i.e. the precision of synchronization will be higher. 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, this approach is quite complex for sensor networks, since 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 figure 3; 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 which yield the best bounds on the estimates- among all data points. The resulting algorithm is much simpler than solving a linear programming problem. However, the authors argue, by a counter example, that this scheme does not always give the optimal solution for the bounds: The algorithm may eliminate some data point, considering it 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, such that it finds the optimal solution with an increase in complexity. The idea is to prevent the algorithm of tiny-sync for eliminating constraints that might be used by some future data points to give tighter bounds. We skip the details here, but the authors basically define a criteria to determine if there is a chance that a constraint might be useful. A constraint is eliminated (discarded) only if it is *definitely useless*. The solutions found by Mini-sync are optimal.

4.4 Lightweight Tree-based Synchronization (LTS)

Lightweight Tree-based Synchronization (LTS), proposed by Greunen and Rabaey [6] 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 needed synchronization accuracy is assumed to be given as a constraint, and the target is to devise a synchronization algorithm with minimal complexity to achieve given precision. This approach is supported by the claim of authors that the maximum time accuracy needed in sensor networks is relatively low (within fractions of a second), and therefore it is sufficient to use a relaxed, or lightweight, synchronization scheme in sensor networks.

Two LTS algorithms are proposed for multihop synchronization of the network based on pairwise

synchronization scheme of [9], as also explained in section 4.2. Both algorithms require nodes to synchronize to some *reference point(s)* such as a sink node in the sensor network. The first algorithm is a centralized algorithm, and needs a spanning tree to be constructed first. Then pairwise 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, and also 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 multihop 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 a 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 of such nodes. On the other hand, letting each node decide on resynchronization may boost the number of pairwise synchronizations, since for each synchronization request, all nodes along the path from reference node to the initiator of resynchronization 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.

5 Conclusion: Remarks and Research Directions

We presented basic synchronization methods proposed for wireless sensor networks, reviewing the motivations and requirements for such work. Two synchronization algorithms, RBS and TPSN, both report very high precisions, on the orders of few μsecs, though they use completely different approaches. The receiver-receiver synchronization of RBS completely eliminates the uncertainty at the sender, and thus is believed by many researchers to perform better than classical sender-receiver synchronization. However, it should be noted that receiver-receiver synchronization requires 4 messages sent and 3 messages received for synchronizing two nodes, while the sender-receiver synchronization requires only 2 sent and 2 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. Mini-sync also relies on a hierarchical structure among sensor nodes, though it is a low-complexity option for synchronizing sensor networks. LTS algorithms offer very low cost synchronization, however with very limited accuracy, therefore with limited applicability. Before concluding the paper, 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 multihop synchronization. When two nodes apart are synchronized using multiple pairwise synchronization steps, the error is usually expected to grow, but since the pairwise errors may have different signs and magnitudes, the overall effect of multihop 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 improve precision, but it also increases consumption of limited resources in the network. The correlation between this redundancy and precision may be investigated. Methods for identifying beacon senders to achieve some desired point in this trade-off curve may be 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, Networked Infomechanical Systems (NIMS) project is a recent initiative towards this, which has already announced the development and deployment of initial prototypes that operated in a forest field biology station[1]. As such systems evolve, synchronization algorithms that take mobility into account will be needed. We believe that synchronization algorithms can even benefit from the mobility, as mobile nodes will "carry" time information from one part of the network to other parts, potentially increasing the global synchronization accuracy.

References

[1] J. Hill, D. Culler, "A Wireless Embedded Sensor Architecture for System-Level Optimization", *Technical Report*, U.C. Berkeley, 2001.

[2] J. Elson and D. Estrin, "Time Synchronization for Wireless Sensor Networks", *International Parallel and Distributed Processing Symposium (IPDPS 2001), Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*, San Francisco, USA, April 2001.

[3] K. Römer. "Time Synchronization in Ad Hoc Networks", *ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 01)*, Long Beach, CA, October 2001.

[4] J. Elson, and K. Römer, "Wireless Sensor Networks: A New Regime For Time Synchronization", *In Proceedings of the First Workshop on Hot Topics In Networks (HotNets-I)*, Princeton, New Jersey, October 2002.

[5] J. Elson, L. Girod, D. Estrin, "Fine-Grained Time Synchronization using Reference Broadcasts", *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[6] J.V. Greunen, and J. Rabaey, "Lightweight Time Synchronization for Sensor Networks", *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA)*, San Diego, CA, September 2003.

[7] A. Hu, and S.D. Servetto, "Asymptotically Optimal Time Synchronization in Dense Sensor Networks", *Proceedings of the 2nd ACM International Conference on Wireless Sensor Networks and Applications (WSNA)*, San Diego, CA, September 2003.

[8] M.L. Sichitiu and C. Veerarittiphan, "Simple, Accurate Time Synchronization for Wireless Sensor Networks", *IEEE Wireless Communications and Networking Conference*, WCNC 2003.

[9] S. Ganeriwal, Ram Kumar, M. Srivastava, "Timing Sync Protocol for Sensor Networks", *ACM SenSys*, Los Angeles, November 2003.

[10] Kay Römer, "Temporal Message Ordering in Wireless Sensor Networks", *IFIP MedHocNet*, Mahdia, Tunisia, June 2003.

---

[1]More information is available online at http://www.cens.ucla.edu/News/ITRgrant.html