Synchronization in Sensor Networks

Blerta Bishaj
Helsinki University of Technology

1. Introduction

The progress in having increasingly smaller and cheaper devices has lead to huge interest in large-scale networks of small, wireless, and low-power sensors. The implementation of these networks ranges from climate studies, military implementations, agriculture, maintenance of machinery, urban disaster prevention, etc.

Time synchronization is critical for distributed systems, but in wireless sensor networks, this is particularly important. Usually, such networks are used for gathering data, and time synchronization is required for reasoning about the events in the physical world. Without a global agreement on time, data from different sensors cannot be matched up. While the requirements for clock accuracy and precision are more severe than in general ad hoc networks, there are energy and channel constraints in sensor networks that make synchronization harder to achieve.

2. Characterizing Time Synchronization

In studying time synchronization in sensor networks, certain metrics are important and vary quite a lot [2].

- *Precision*. The maximum error in relation to a standard.
- *Lifetime*. It can range from synchronization that lasts as long as the network operates, to almost momentary (for example, for comparing the detection time of a single event).
- *Scope and Availability*. The geographic span of the synchronized nodes, and the completeness of coverage.
- *Efficiency*. The time and energy needed for the synchronization.
- *Cost and Form*. Some sensor networks might need small, low-cost, disposable nodes.

Sensor networks are used in situations that measure differently according to the above metrics. A sensor network cannot be optimal in all them. For example, GPS receivers' synchronization can be persistent, with a precision of 200ns. However, they cannot be used inside buildings, underwater, on Mars. They might also be too large, power-consuming, or expensive in some cases.

On the other hand, let's consider a group of small, short-range nodes. One node can transmit a signal, which the others can use as a time reference. The synchronization provided in this case is local in scope, limited in precision because of the propagation delay of the radio waves. However, this synchronization is fast and energy-efficient.

Something to note is that the granularity of the clock affects power consumption. High frequency clocks consume more, and they should be used if the particular task really requires such precision. To save energy, we might apply post-facto synchronization protocols [5], if the purpose of the network does not require all-time synchronization. In this scheme, nodes go to a low-power state, with unsynchronized clocks. Then, after an event of interest happens, the clocks are synchronized. This method eliminates unnecessary synchronization.

3. Causes of clock desynchronization

Clocks have some (quartz) crystal that vibrates at a certain frequency when electricity is applied to it. The reasons that two clocks might be desynchronized are [6]:

- *Offset*. The clocks may have been started at different times.
- *Skew*. The clocks might be running at slightly different frequencies, causing them to diverge over time.
- *Drift*. The frequency of the clocks can vary over time. This is called the drift error. It can be subdivided into *short-term* and *long-term* frequency drift. The short-term one is due to environmental factors: temperatures, shock, supply voltage. The long-term one is due to the aging of the oscillator (crystal properties).

Nodes in a sensor network may not have been synchronized well initially, or the clock may drift due to temperature, pressure, shock, battery, etc. Sensor nodes are particularly prone to such conditions that would cause frequency differences in the oscillators, and thus, phase differences between the nodes.

4. Algorithms

This section contains an overview of several algorithms for time synchronization in sensor networks [3].

All-node-based synchronization

This section describes a method for global synchronization of clocks in a sensor network. We assume that the clock cycles of all nodes are the same. We also assume that the clock tick time is longer that packet transmission time, because small clock tick times require high frequency, which requires energy.

This algorithm assumes the message transmission time and packet handling time at each node is the same. This assumption holds, because a sensor network can suppress all other messages except synchronizing ones when the network is being deployed. The idea behind the algorithm is to send a message in a loop, record the initial and the end time of the message. Afterwards, we can average the time required for each segment, and regulate the clocks accordingly.

The algorithm times the routing of a message along specified paths, and uses the time differences to correct the times for the nodes in the paths. The synchronization is divided into two phases. In the first one, a synchronization packet is sent along a cycle. The node that initiates it, records the starting and ending time of the packet with its own local clock. The other intermediary nodes forward the packet, and record how many hops the packet has done so far. In the second phase, the initiating node sends a correction packet along the same loop. This packet contains the starting and the ending time of the first packet, as well as the total hops in the cycle. Each node, then, computes how to adjust its clock. The figure below depicts how the synchronization is made.

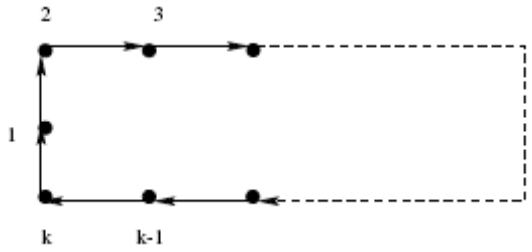

Figure 1: A clock synchronization message traveling along a loop originated at node n_1 and then back to node n_1 [3].

It can be proved that this method constrains the relative clock error to at most three times the clock cycle.

Cluster-based synchronization

The all-nodes-based algorithm has the disadvantage of requiring that all nodes take part in one single synchronization session. This can prove hard to achieve. Instead, a hierarchical approach can solve this. If the nodes can be organized into clusters, then we could have two rounds of synchronization. In the first one, the cluster heads synchronize in one loop. Then, in the second round, each cluster head initiates synchronization within the cluster. Each round uses the all-nodes-based algorithm, for a group of nodes. Extending this algorithm to hierarchical levels increases the flexibility and scalability of the algorithm, but the down side is that it decreases the precision of the synchronization. Each round has a maximal error of three times the clock cycle. With two levels of hierarchy, we end up with a maximum error of six times the clock cycle.

Synchronous diffusion

Why is diffusion needed?

The previous method uses global time information that is sent to all nodes, and they are not scalable for very large networks. Furthermore, the initiation node may experience failure, thus the methods are not fault-tolerant. Also, the algorithm of the method requires to be executed at approximately the same time, but this may be difficult to attain in large systems.

The Rate-based Synchronous Diffusion Algorithm

Let us suppose that we have n sensors in the system. We can describe the network as a graph G(V,E), in which the vertices are the sensors, and the edges indicate the connectivity between the sensor nodes. Let $C = (c_1^t, c_2^t, \cdots, c_n^t)^T$ be the time readings of the sensors i = 1 .. n, at time t. If n_i and n_j are within each-other's transmission range and $c_i > c_j$, then we certainly want to decrease c_i and increase c_j. Because we are trying to synchronize the clock for all the sensors in the system, the decrease of c_i should go to the increase of c_j. Let us suppose that the diffusion value is proportional to $c_i - c_j$, and that the diffusion rate is $r_{ij} > 0$ ($r_{ij} = 0$ if n_i and n_j are not neighbors). The r_{ij} values are such that $\Sigma_{j<>i} r_{ij} <= 1$.
The algorithm that shows this diffusion method is:

Algorithm: Diffusion algorithm to synchronize the whole network

1: Do the following with some given frequency
2: **for** each sensor n_i in the network **do**
3: Exchange clock times with n_i's neighbors
4: **for** each neighbor n_j **do**
5: Let the time difference between n_i and n_j be $t_i - t_j$
6: Change n_i's time to $t_i - r_{ij}(t_i - t_j)$

As can be seen, the exchanged time value between two neighbors is proportional to the time difference between them. In this algorithm there is no absolute clock time value; there is no requirement for all the sensor nodes to perform this local synchronization at the same time. It can be proved that this algorithm converges.

The Asynchronous Diffusion Algorithm

The previous algorithm has the disadvantage of being synchronous: it requires things to be done in order. A node cannot be interrupted before finishing its current round of operations. The asynchronous diffusion algorithm dismisses this constraint, provided that each node gets involved in the operations with non-zero probability.

The Asynchronous Averaging Algorithm is based on an average operation of a node with its neighbors. Each node asks its neighbors about their clock values, computes the average, and then sends it out so that they can update their values. The assumption is that a node takes part in such exchanges in at most one neighborhood at one time. It can be proved that the algorithm converges to the global average value.

Reference Broadcast Synchronization (RBS)

Precise network synchronization is hampered by non-determinism. The estimates for latency are confounded by events that last for an amount of time that we can not determine. If we decompose the sources of message latency, we would have these components [5]:

- *Send time*. This is the time spent at the sender to construct the message and to transfer it to the network interface.
- *Access time*. The delay incurred by the MAC protocol, for gaining access to the transmit channel, for example several control packet must be exchanged at the MAC layer before data can be sent. The MAC delay also depends on how many nodes there are in the network, and how many of them are sending messages currently.
- *Propagation delay*. The time needed for the message to reach the destination, once it has left the sender. This time is small when communicating nodes are within the reach of each-other. On the other hand, it is big if other nodes serve as intermediaries, because then time is spent for packet queuing and processing.
- *Receive time*. Processing required for the packet at the receiver. If the arrival time of the packet is timestamped in a low enough layer of the receiver's stack, the overhead of packet processing is eliminated from the computations.

According to [5], in RBS, a message is broadcast to two receivers, which later synchronize their local clocks together. Each of the receivers records and exchanges the local time when the message was received.

The advantage of RBS is the elimination of non-determinism on the transmitter side; it eliminates the *Send time*, and *Access time* from the equations. The assumption of RBS is that *Propagation time* is 0, since RBS is based on broadcast and there are no intermediary nodes. As for *Receive time*, as already mentioned, early timestamping eliminates it.

The precision of the estimation can be improved if, instead of one message, m messages are sent:

- the transmitter broadcasts m reference messages
- each receiver records the local time of interception of each message
- receivers exchange their observations (these local times)
- each receiver can compute its offset to any other receiver as the average of the offsets for m messages

With m messages being sent, the RBS synchronization can be improved to take the clock skew into consideration. One proposal for doing that is the least-squares linear regression method. This method can be used when m messages are sent in RBS. More on this method can be found in [7].

Multi-hop synchronization

RBS offers local synchronization, but it can be extended to provide global synchronization. It can be provided with multi-hop synchronization.

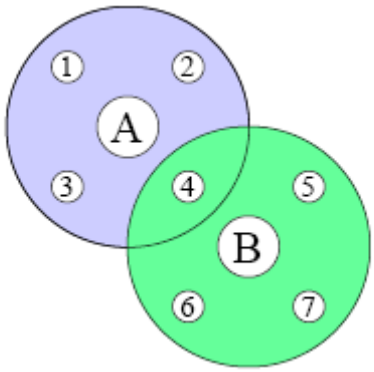

Figure 2: A topology where multi-hop synchronization is required [5].

As can be seen in the figure above, A and B cannot hear each-other, but they are both heard by node 4. Node 4 can relate the clocks in one neighborhood to clocks in the other.

Time-sync Protocol for Sensor Networks (TPSN)

TPSN is based on building a tree of the sensor network [4]. The tree starts from a root, and expands further. Each node establishes its level by the level-discovery message it receives. As the tree is being built, pair-wise synchronization takes place along the edges between each node and its parent. Let us have two nodes: N_2 and its parent N_1. When N_2 wants to synchronize, it sends a synchronization pulse to N_1, with the time T_1 of sending

from N_2. The parent, after receiving the packet, sends back an ACK packet with the reception time T_2, and the retransmission time T_3. N_2 receives the packet at time T_4; it can use these four time values to find the offset and the propagation delays, accordingly.

$$\Delta = ((T_2 - T_1) - (T_4 - T_3)) / 2$$

$$d = ((T_2 - T_1) + (T_4 - T_3)) / 2$$

TPSN is classified a sender-initiated synchronization protocol, where the sender synchronizes to the receiver's clock. Dynamic topology changes affect TPSN very much, as the tree has to be rebuilt, and time synchronization has to be done with the new parents.

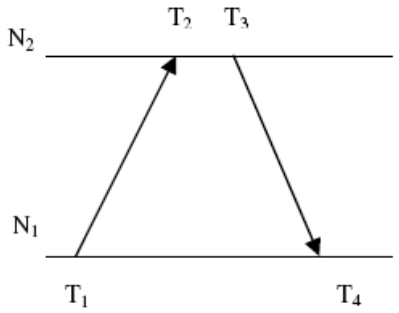

Figure 3: An example of TPSN [4].

Flooding Time Synchronization Protocol (FTSP)

In FTSP [4], a root node broadcasts its local time, and any receiving node synchronizes its clock to that time. Each such message contains: *rootID*, *seqNum*, *sendingTime* (global time of the sender at transmission time). When a node receives such a message, it calculates the offset of its time. If a node has a set of these values, it can calculate its clock skew using linear regression [7].

FTSP can also provide multi-hop time synchronization: A node, upon receiving a message from a root node, updates its global time. Additionally, it broadcasts its own time to its neighbors. All nodes should behave this way. However, to avoid using redundant messages in the linear regression method, a message is used for this method only if the *seqNum* field is bigger than the biggest one received so far, and if the *rootID* of the message (the origin of the flood) is not bigger than the last received *rootID*.

FTSP is more robust than TPSN, because there is no topology defined and it can adapt to the failure of a node. If a node does not hear a time synchronization message for a specified period, it declares itself root. To ensure that there is only one root, a root gives up being so if it hears a message with a smaller ID.

5. Time synchronization attacks in sensor networks

As already explained earlier in this paper, time synchronization is crucial in sensor networks. The problem is that protocols for time synchronization have not been designed with security in mind [4]. If an adversary could compromise a node, the network might not be able to succeed in its mission. The functioning of the network might even be compromised by disrupting, for example, the TDMA-based channel scheme.

Attacks on RBS

Just as a reminder, a reference message is broadcast by a node, and is used by two other nodes for synchronization. But, if a node is compromised, it will send wrong synch information to the other node. As a result, the honest and unsuspicious node will calculate a wrong offset and skew. The whole network can become desynchronized, as a result. This is explained in the field of Robust Estimation, and the *breakdown point* of an estimator – the smallest fraction of contamination in the data that causes the estimated values to be way wrong. If the average or the linear regression methods are used, for big networks and as the number of nodes increases, even a small fraction of the data being wrong is enough for the estimation to diverge. Methods other than linear regression have been proposed that have a high breakdown point. Therefore, they are more resilient to false information.

The multi-hop version of RBS can also be attacked. RBS does not keep a global time, but it uses nodes on the boundary of overlapping regions to global synchronization. If such a node is malicious, it can desynchronize the clocks of other nodes, or compromise the result of the observations.

Attacks on TPSN

As mentioned before, TPSN is sender-based; therefore, a malicious node will only be able to affect its children. It can send them incorrect timestamps. The protocol causes the error to be propagated down the branch that is headed by the malicious node. The closer to the root this node is, the more nodes are contaminated.

A malicious node can also claim a lower level, making other nodes request synchronization from it. Additionally, if a node did not participate in the tree building and its branch nodes had no other way of connecting to the tree, they would become disconnected.

Attacks on FTSP

FTSP has the advantage that the root is chosen dynamically. One possible attack is that a malicious node can claim to be the root if it starts transmitting messages with ID 0, and a higher sequence number than the real root is transmitting. This is very easy to do in this protocol, it uses the mechanism of this protocol to handle the case when the root either does not exist, or is unheard from for some time. After becoming the root, it is very easy to, for example, modify the sending time in the message, and the synchronization becomes faulty.

6. Conclusions

Time-synchronization protocols are crucial to sensor networks, such importance being driven by the very nature and purpose of these networks. Specific protocols have to be designed that take into account the mobility, the energy-constraints, the other underlying protocols, etc. Something to bear in mind is that higher accuracy costs in: more clock ticks, more message exchanges and processing, more computations of the results, which sensor nodes can hardly afford. Therefore, there is a trade-off to be made. Something else that is also very important, but often neglected, is security. These protocols have to

handle the possibility of malicious nodes, because synchronization is very easy to compromise.

7. References

[1] Jeremy Eric Elson, "Time Synchronization in Wireless Sensor Networks", 2003

[2] Jeremy Elson and Deborah Estrin, "Time Synchronization for Wireless Sensor Networks", 2001

[3] Qun Li and Daniela Rus, "Global Clock Synchronization in Sensor Networks", 2004

[4] Michael Manzo, Tanya Roosta and Shankar Sastry, "Time Synchronization Attacks in Sensor Networks", 2005

[5] Jeremy Elson, Lewis Girod and Deborah Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts", 2002

[6] Saurabh Ganeriwal, Srdjan Capkun, Chih-Chieh Han, Mani B. Srivastava, "Secure Time Synchronization Service for Sensor Networks", 2005

[7] http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm, 24 April 2007