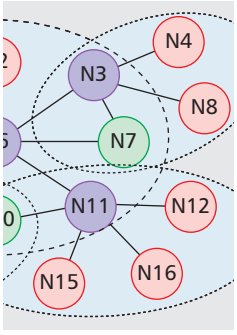
# Synchronization of Multihop Wireless Sensor Networks at the Application Layer

Álvaro Marco and Roberto Casas, University of Zaragoza
José Luis Sevillano Ramos, University of Seville
Victorián Coarasa and Ángel Asensio, University of Zaragoza
Mohammad S. Obaidat, Monmouth University




The authors present a method that allows accurate synchronization of large multi-hop networks, working at the application layer while keeping the message exchange to the minimum.

## Abstract

Time synchronization is a key issue in wireless sensor networks; timestamping collected data, tasks scheduling, and efficient communications are just some applications. From all the existing techniques to achieve synchronization, those based on precisely time-stamping sync messages are the most accurate. However, working with standard protocols such as Bluetooth or ZigBee usually prevents the user from accessing lower layers and consequently reduces accuracy. A receiver-to-receiver schema improves timestamping performance because it eliminates the largest non-deterministic error at the sender's side: the medium access time. Nevertheless, utilization of existing methods in multihop networks is not feasible since the amount of extra traffic required is excessive. In this article, we present a method that allows accurate synchronization of large multihop networks, working at the application layer while keeping the message exchange to a minimum. Through an extensive experimental study, we evaluate the protocol's performance and discuss the factors that influence synchronization accuracy the most.

## Introduction

Time synchronization entails an important function in wireless sensor networks (WSNs), and this function can be performed at different layers depending on the objective of synchronization. For instance, sharp timing is fundamental at low layers as it helps increase data rates (short bit times), and enhance noise immunity (frequency hopping) and time-division multiple access (TDMA)-based scheduling. Furthermore, since radio is usually the most energy-consuming part of a node, keeping the nodes awake the minimum required time to exchange data is a common practice that requires synchronization.

Synchronization is also of great interest at higher layers. Akyildiz *et al.* identify in the application layer the *sensor management protocol*, which includes (among other administrative tasks) time synchronization [1]. From a practical point of view, WSNs are networks composed of a large number of small devices that take measurements, process them, and communicate with other devices coordinating their operations. This collaboration enables a complex sensing task, named *data fusion* [2]. Data fusion requires synchronization for two tasks: time scheduling and timestamping. The first is needed when the nodes coordinate to perform cooperative communications. The second is commonly used when data is fused taking into account the collecting instant; for example, to perform event detection, tracking, reconstruction of a system's state for control algorithms, and offline analysis.

In this article we describe in detail and evaluate the use in WSNs of the Multihop Broadcast Synchronization (MBS) protocol. An early version of this scheme was first introduced by us in [3]. The proposed protocol is very well suited for WSNs and fills an existing gap when synchronizing WSNs with a global network time. As seen in the following sections, MBS helps to achieve high accuracy and energy efficiency when timestamping at lower layers is not possible in multihop networks.

## Related Work

Creating a common temporal reference using the nodes communication capabilities has been widely studied [2]. According to the strategy, we could distinguish between a posteriori and a priori synchronization [4]. A posteriori methods keep devices' clocks running free, gathering information between relative clocks and rearranging timestamps once the measurement processes are finished. These methods are usually the most energy-efficient because they optimize the number of messages exchanged, but they do not offer real-time capabilities. On the other hand, a priori methods overcome this by synchronizing all the nodes with a common time reference (global network time [GNT]) using regular clock corrections. A common drawback of these techniques is overload of the network

due to the messages required to estimate the communication delays.

Another key issue is whether there is a sender transmitting the current clock values as timestamps (*sender-to-receiver*) or not (*receiver-to-receiver*). According to the time analysis performed by Maróti *et al.*, the most significant delays when transmitting messages over a wireless link are those from the send, receive, and access processes [5]. The problem with sender-to-receiver methods is the uncertainty time introduced by the send and access processes. As a receiver-to-receiver method, the Reference Broadcast Synchronization (RBS) protocol proposes the use of reference broadcast messages to establish a common time reference and get rid of the transmitter-side non-deterministic error sources (it is assumed that all devices listening to the broadcast get the message at the same time). This eliminates the time uncertainty introduced by the send and access processes, and sets a temporal reference shared by all the nodes [4].

The biggest drawback of receiver-to-receiver synchronization methods is how to propagate the local timestamps of the broadcast-receivers to set a GNT. Elson and Estrin propose a post facto synchronization method, that is, a method that performs synchronization only when it is needed [6]. The scattering method they propose does not give a common time reference to the broadcast sender; it only synchronizes receivers. However, various authors point out the need of setting a network time to propagate the synchronism over a multihop network using broadcasts [4, 7]. Thus, nodes in the broadcast domain need to share timing information among them to determine the GNT.

On the other hand, Timing-sync Protocol for Sensor Networks (TPSN) is a sender-to-receiver protocol that achieves real time with high accuracy optimizing message exchange. It avoids the indeterminism working at the medium access control (MAC) layer to precisely timestamp messages at the exact moment they are sent [8]. The Flooding Time Synchronization Protocol (FTSP) also uses MAC layer timestamping at both the sender and receiver sides. The protocol proposes a multihop propagation scheme that does not need any initial configuration to propagate synchronization info. This ad hoc structure also enables dynamically overcoming node and link failures in the network [5].

However, these sender-to-receiver synchronization schemes require accessing the lower layers, which is not always possible when using standard or complex protocols. Current WSN applications are developed using a wide variety of hardware, software, and communication protocols. Some of them are proprietary; others use common development platforms that have become de facto standards (Motes, i-beans); and others implement the two current standards suitable to be used in WSNs: ZigBee and Bluetooth. In all these cases accessing the lower layers is not possible, and this prevent us from using sender-to-receiver synchronization.

The Multihop Broadcast Synchronization protocol is a receiver-to-receiver synchronization scheme that nonetheless obtains a GNT working at the application layer. MBS is suitable for large multihop networks, keeping the number of messages in the same order of magnitude when compared to the sender-to-receiver methods.

## Multihop Broadcast Synchronization Protocol

Many of the functions performed by sensor nodes require a precise time measurement (e.g., bit time calculation in communications). The devices commonly used to provide an accurate time base are oscillators. Although these clocks ideally provide a heartbeat at a constant rate, all of them present a frequency tolerance (whose limits are provided by its manufacturers) that is also slightly affected by the operation temperature and aging. Thus, two clocks, even manufactured in the same process, will not have a priori the same frequency.

This way, nodes in a WSN measure time with oscillators at slightly different rates. This divergence, called clock skew, can go up to 150 parts per million (PPM) (i.e., each second the nodes will commit an error than can go to 150 μs). Apart from the clock skew, there is an offset among clocks because each node starts at a different instant. Given a node $i$, with clock skew $s_i$ and offset $k_i$, its clock reading $t_i$ can be written as

$$t_i = s_i t + k_i, \; i = 1 \ldots n. \tag{1}$$

Thus, synchronizing the clock of node $i$ implies estimating and compensating clock skew and offset $(s_i, k_i)$. The most used procedure to perform this adjustments is broadly described in the literature [2, 4, 5, 9]. There is a reference clock $t_r$ to which all the nodes are synchronized. A sync-point $k$ is defined as a pair of timestamps collected at the same time $t^k$ in the reference node and in the nodes that want to be synchronized: $\{t_i^k, t_r^k\}$. Once each node stores several sync-points at different instants, the offset and skew $(k_i^*, s_i^*)$ differences with the reference can be calculated using linear regression:

$$\begin{aligned} s_i^* &= \frac{\sum_k \left(t_r^k - \bar{t_r}\right)\left(t_i^k - \bar{t_i}\right)}{\sum_k \left(t_r^k - \bar{t_r}\right)^2} \\ k_i^* &= \bar{t_i} - s_i^* \bar{t_r}, \end{aligned} \tag{2}$$

where the bar indicates average. This way, every node can estimate the global time $(t_r^*)$ from its local clock:

$$t_r^* = \frac{t_i - k_i^*}{s_i^*}. \tag{3}$$

Once a node is synchronized, it can propagate the estimated $t_r^*$ to others, creating new sync-points that spread the GNT in multihop networks [5].

Strictly speaking, it is not possible to obtain a sync-point at the same instant in two nodes not physically linked; there will be unknown delays in the synchronization message exchange. While deterministic uncertainties in wireless links slightly affect the synchronization accuracy, non-deterministic ones drastically reduce it [1, 4, 5,

Although these clocks ideally provide a heartbeat at a constant rate, all of them present a frequency tolerance that is also slightly affected by the operation temperature and aging. Thus, two clocks, even manufactured in the same process, will not have a priori the same frequency.

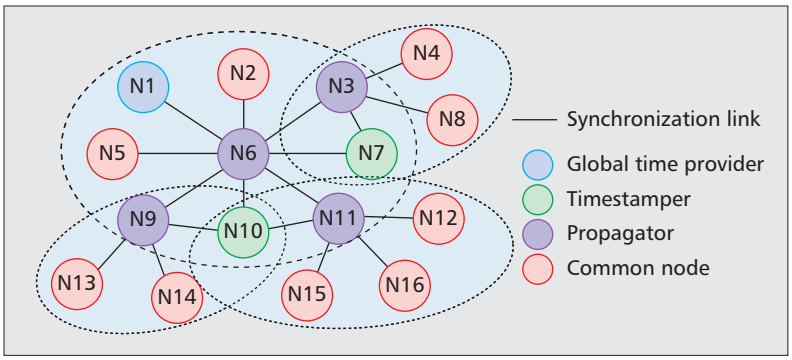



**Figure 1.** *A grid network where propagator nodes send broadcast messages, and time stamper nodes reply to them with the arrival time of the broadcast. All the nodes also have direct communication links with their neighbors.*

8]. TPSN and FTSP eliminate the biggest uncertainties (send, receive, and access time) by timestamping at the MAC layer the send and receive instant of a message. This way, to obtain the required sync-points accurately, only one message is necessary.

When access to the low layers is not possible, reference broadcast messages eliminate the uncertainty at the sender side. Unfortunately, although the reception instant of a broadcast message is tight, this procedure does not provide the required sync-points directly because the sender is not synchronized [4, 10]. To the best of our knowledge, existing methods require additional message exchange between every receiver node to set the GNT. This makes multihop propagation very inefficient [4, 7].

Our MBS protocol obtains sync-points using reference broadcasts with considerably less message overhead than other methods that also use reference broadcasts, and manages to synchronize all the nodes (even the broadcast sender) as well, making multihop propagation easy.

## Protocol Description

To illustrate how MBS works, we consider the example network in Fig. 1. Nodes in MBS perform two different tasks. *Propagators* are those that spread the GNT broadcasting synchronization messages. *Timestampers* are nodes that can notify the propagators about the timestamp when the previous broadcast message arrives. In Fig. 1, N3, N6, N9, and N11 are propagators. N1, N7, and N10 are timestampers; the provided timestamps are referred to the GNT, so they have to be synchronized when notifying the corresponding propagator node. To overcome this requirement when initiating the process, N1 will be the node whose local clock will be the GNT (i.e., N1 is the global time provider [GTP]).

The synchronization will be made in two hops, the same number of hops FTSP would need. In the first hop, N6 will synchronize the nodes in the dashed ring to N1's clock (using a technique similar to that of RBS). Then, N3 using N7 as timestamper, and N9 and N11 using N10, will synchronize the nodes in the dotted rings and propagate the GNT one hop away. N6 must also be synchronized, and, as it is the only propagator connected to the GNT, it will be synchronized in the second hop by any of the other propagators.

This process will be done using two different messages: *SyncBC* and *TimeUC*. The first type is broadcast by propagators and is equivalent to the reference broadcast in RBS: messages that trigger timestamping of the receiving instant at the sender side. It contains three fields: the *propagatorID* identifying the sender, the *sequenceNumber* of the message, and the *timeStamp* when the previous *SyncBC* arrived. These messages are used to propagate the GNT the same way as synchronization messages in FTSP [5]. *TimeUC* messages are unicast messages used by timestampers to notify the propagators about the time when the last *SyncBC* arrived. They have the following fields: the *timeStamperID* identifying the sender, and the corresponding *propagatorID*, *sequenceNumber*, and *timeStamp* about which they are informing. Now we explain the sequence to synchronize the network in Fig. 1. We indicate the message sent specifying the fields: *SyncBC* (*propagatorID*; *sequenceNumber*; *timeStamp*) and *timeUC* (*timeStamperID*; *propagatorID*; *sequenceNumber*; *timeStamp*). The first hop would be as follows:

1. N6 initiates the synchronization process by sending a *SyncBC* (*N6*; *0*; *void*) message. The nodes that receive the message (N1, N2, …) timestamp the arrival of the *SyncBC* from node 6 with sequence number 0; that is, $TS_{N1}\{N6, 0\}$, $TS_{N2}\{N6, 0\}$, ….
2. N1 informs N6 about the timestamp when it received the last *SyncBC* message: *TimeUC* (*N1*; *N6*; *0*; $TS_{N1}\{N6, 0\}$).
3. N6 sends the timestamp of the previous *SyncBC* message and sets a new reference point for timestamping: *SyncBC* (*N6*; *1*; $TS_{N1}\{N6, 0\}$). At this instant, all *Ni* neighbors of N6 have their respective sync-points from the first *SyncBC*: [$TS_{Ni}\{N6, 0\}$, $TS_{N1}\{N6, 0\}$]. N1 does not need it because it rules the GNT.

From now on, steps 2 and 3 will be repeated, causing all nodes in range of N6 to have a collection of sync-points. Then, using linear regression, they get synchronized, calculating their offset and skew differences to the reference clock. In that first hop, all nodes within the dashed ring will be synchronized to N1's clock.

Note that N6 does not have the global time. To fix this and to propagate the clock one hop away, any other propagator, such as N3, initiates the above described sequence. The subsequent hops will be performed following the same procedure. Each propagator broadcasts *SyncBC* messages including the timestamp of the previous *SyncBC* message. Timestampers notify the propagators about the last timestamp. All nodes in a range get a collection of sync-points that allow them to synchronize to GNT.

Accuracy of the sync-points will be degraded as nodes are farther away from the clock generator (N1), similar to sender-to-receiver methods. When synchronizing the nodes situated within the dotted rings, the timestampers used by the propagators (N7 and N10) are one hop away from the GNT (N1), which will increase the error committed.

In case of near failure (i.e., low batteries), propagators and timestampers can transfer their responsibilities to neighboring nodes. In any case, similar to the scheme proposed by Maróti

*et al*., a network could be autonomous and self-healing in terms of synchronization using node identifiers to automatically assign roles [5].

### Practical Issues

Earlier we described the MBS protocol without taking into account practical issues like how to initialize the algorithm, how to determine the role to be performed by every node, and so on. To understand the general case, let us first consider several examples.

Let us assume that we have a node, N1, that is already synchronized, and a couple of nodes, N2 and N3, that are not yet synchronized. In order to synchronize N2 and N3 with respect to N1, an intermediate node, N4, that can send broadcast messages to N1, N2, and N3 is needed. Thus, N4 will act as a propagator node to its neighbors, and N1 will act as the timestamper for N4. Thus, the general rule is that *all the neighbors of the neighbors of N1* can be synchronized with respect to N1 provided that these nodes have a one-hop estimation of the time of N1. In the same manner, N1 will have a one-hop estimation of the time in the node with respect to which it is synchronized, which in turn will have a one-hop estimation, and so on until the GTP is reached. The number of hops needed to reach the GTP can be used to define the quality of a node synchronization, which we call *HopId*.

Now consider a node N1 that is not synchronized. In order to become synchronized, a nearby — synchronized — node must assume the role of timestamper. Analogous to the previous case, all the neighbors of the neighbors of N1, which are already synchronized, can behave as timestampers. Therefore, the one with the best GNT estimation (i.e., the lowest HopId) must be chosen as timestamper. If more than one node have the lowest HopId, the one that has more *synchronizable* nodes will be chosen, and the node that is neighbor of both the timestamper and N1 will act as the propagator.

In order to improve performance, if a node listens to more than one propagator, it must only use sync messages from the propagator whose timestamper has lower HopId. Also, if a node is acting as timestamper for a propagator, it cannot use incoming *SyncBC* messages from this propagator to synchronize itself.

Now, the mechanism should be clear. First, all nodes but the GTP (with HopId equal to 0) are not synchronized. Nodes close to the GTP will be synchronized with respect to the GTP, and they will be assigned a HopId equal to 1. Synchronization will spread across the entire network following the rules discussed above.

Still, several criteria are possible to set the GTP, such as minimizing the total synchronization error expressed as the sum of the HopId of every node, maximizing the number of nodes with lower HopId, or setting an upper bound for the HopId.

## MBS Protocol Evaluation and Comparison

We have evaluated the MBS protocol in two different architectures. Both are used to build multihop WSNs following two standard protocols: ZigBee and Bluetooth. In the case of ZigBee, we use a common platform: an Atmel microcontroller (ATMega128) with the Chipcon's CC2420 transceiver [9]. Developing the entire stack to make the network ZigBee-compliant requires a lot of work, thus we used the EmberZNet embedded software. This way, we built a multihop, auto-routing and self-healing ZigBee network working at the application layer.

For the Bluetooth architecture, a Microchip PIC16F876 microcontroller manages a Mitsumi Bluetooth module (WML-C20) through HCI, the standard Bluetooth Host Controller Interface. This design enables us to access low-power modes and implement tree topology networks using scatternets.

To evaluate MBS's performance, we connect every node to a wired bus, where one of them periodically generates a pulse. All the nodes, which are synchronized with MBS, timestamp the receiving instant with their GNT estimation and send back the data to the PC, where the timestamp differences with respect to the GTP are considered to obtain the synchronization error.

### Single-Hop Synchronization

When synchronizing wireless nodes, all methods use one of the following strategies: to timestamp at the sender and receiver side, or use reference broadcasts, timestamping only at the arriving side. In Table 1 we compare the alignment errors of some synchronization schemes presented in the references.

The timing accuracy among nodes depends mainly on the hardware and firmware architecture: how the sending and arriving moments are detected, and which times (propagation, access, etc.) are affected and their uncertainty. Errors shown in Table 1 determine the accuracy of each sync-point that will be used to perform the linear regression in Eq. 2. Of course, the fewer the errors, the more accurate the synchronization. Other factors that also affect the estimation are the following:

- The distribution of the errors (uniform, Gaussian, etc.) will determine how the linear regression eliminates them and the quality of the clock estimation. The time difference between reception instants of broadcast messages follows a Gaussian distribution with the architectures described before [3, 4].
- The local oscillator drift is influenced by the initial accuracy (difference between the oscillator output frequency and the specified frequency at 25°C at the time of shipment by the manufacturer), temperature stability, and aging. Its behavior can also condition the precision and the timing lifetime.
- Finally, the frequency and number of sync-points used in the estimation will determine the expected accuracy.

As stated by van Greunen and Rabaey, not all sensor networks applications have the same sync needs in terms of accuracy [11]. In order to provide the reader with some guidelines that could help in deciding on the best suited hardware (transceivers, crystals, etc.) and firmware (sync-message rate, number of data points to perform regression) in each application, we have characterized synchronization behavior in several

All the nodes, which are synchronized with MBS, timestamp the receiving instant with their GNT estimation and send back the data to the PC, where the timestamp differences with respect to the GTP are considered to obtain the synchronization error.

| | Average error (μs) | Worst case error (μs) |
|---|---|---|
| **Sender — Receiver synchronization** | | |
| ZigBee (Motes 2.4 GHz) [9] | 14.9 | 61.0 |
| TPSN (Motes 916 MHz) [8] | 16.9 | 44.0 |
| FTSP (Motes 433 MHz) [5] | 1.4 | 4.2 |
| **Receiver — Receiver synchronization** | | |
| RBS (Motes) [4] | 21.9 | 93.0 |
| MBS (Bluetooth) | 4.5 | 18.0 |
| MBS (ZigBee) | 22.2 | 52.0 |

**Table 1.** *Comparison of alignment error in synchronization methods.*

scenarios. We compared the two architectures described above, Bluetooth and ZigBee, each with different alignment errors, as shown in Table 1. Both alternatives have been tested with two local oscillators with different frequency stabilities of 40 PPM and 150 PPM. We have also changed the synchronization rates (30 s and 300 s) and number of sync-points used in the regression (3, 6, 8, 12, 20, and 50). In Table 2 we show the average and maximum synchronization error with 95 percent probability for both scenarios.

The first interesting result is that synchronization precision depends mainly on the number and accuracy of the sync-points used to estimate parameters by regression. Thus, if we have very low computation resources or need low-accuracy timing we can use a few sync-points (i.e., a lightweight method) [11]. In contrast, if the highest accuracy is needed, we need powerful hardware to use the maximum number of points in the regression.

Contrary to what people might expect and agreeing with what Maróti *et al*. have found out, synchronization rate and oscillator accuracy barely affect precision [5]. This makes sense if we consider the local clock stable in the short to medium term, something that fortunately is so in most cases. Aging has a negligible effect on stability: one to three PPM each year. Temperature influences clock drift more severely: tens of PPMs in the operating temperature range. In worst cases, this translates into an error of 1 μs/°C.

The reduced influence of synchronization rate can be used in many ways. By increasing it, we can reduce the initial settling time in multihop networks, quickly synchronize a new node, or maintain accuracy in sudden temperature variations. Lowering the rate will be useful to minimize extra traffic and reduce power consumption. Despite the results shown in Table 2, the synchronization interval cannot be as long as we want. Linear regression estimates the skew and offset of the local clock referred to the GNT, and there will always be errors.

According to Eq. 1, the more time from the last resynchronization, the larger the error of

| | | ZigBee 40 PPM | | ZigBee 150 PPM | | Bluetooth 40 PPM | | Bluetooth 150 PPM | |
|---|---|---|---|---|---|---|---|---|---|
| | | Avg. | Max.[a] | Avg. | Max.[a] | Avg. | Max.[a] | Avg. | Max.[a] |
| $N = 3$ | $tSync = 30$ s | 39.26 | 105.20 | 39.45 | 106.42 | 7.67 | 20.92 | 8.10 | 21.70 |
| | $tSync = 300$ s | 39.99 | 108.09 | 39.59 | 109.05 | 8.00 | 21.61 | 7.73 | 20.88 |
| $N = 6$ | $tSync = 30$ s | 21.12 | 52.65 | 22.06 | 55.23 | 4.17 | 10.30 | 4.34 | 10.77 |
| | $tSync = 300$ s | 21.90 | 53.52 | 21.29 | 52.07 | 4.41 | 10.99 | 4.36 | 10.93 |
| $N = 9$ | $tSync = 30$ s | 17.27 | 40.95 | 16.58 | 40.66 | 3.30 | 8.19 | 3.30 | 8.25 |
| | $tSync = 300$ s | 16.10 | 39.38 | 15.72 | 38.90 | 3.28 | 8.22 | 3.36 | 8.39 |
| $N = 12$ | $tSync = 30$ s | 13.33 | 32.83 | 14.39 | 35.13 | 2.82 | 6.84 | 2.79 | 6.89 |
| | $tSync = 300$ s | 13.99 | 34.52 | 13.77 | 34.87 | 2.73 | 6.62 | 2.68 | 6.63 |
| $N = 20$ | $tSync = 30$ s | 11.29 | 27.20 | 9.83 | 24.23 | 1.96 | 4.71 | 2.13 | 5.14 |
| | $tSync = 300$ s | 10.61 | 26.13 | 10.46 | 25.51 | 2.19 | 5.29 | 2.13 | 5.30 |
| $N = 50$ | $tSync = 30$ s | 7.03 | 17.46 | 6.77 | 15.58 | 1.23 | 2.93 | 1.27 | 3.10 |
| | $tSync = 300$ s | 6.22 | 14.80 | 7.54 | 18.43 | 1.30 | 3.27 | 1.32 | 3.21 |

*N* is the number of sync-points used to perform regression, and *tSync* is the synchronization interval.
[a] Maximum synchronization error with 95% probability.

**Table 2.** *Synchronization error in the MBS method with one hop (in μs).*

this estimation. Figure 2 shows the synchronization error between two Bluetooth nodes and the reference node. Here we used sync messages every 30 s, and after 500 messages (about 4 h) the synchronization process was stopped. We can see how after the stopping instant, the synchronization error of each node grows depending on the last estimation of the skew and offset. Additional results may be found in [3].

## Multihop GNT Propagation

Cumulative errors when propagating the network time only depend on the estimation's accuracy of the corresponding timestamper's clock. When the timestamper is the one setting the reference, the precision will be as described earlier. Differences arise when it owns an *n*-hop estimation of the network clock.

We have evaluated the behavior of MBS in two different scenarios. With the same philosophy of the single-hop case, we have tested its performance with different numbers of sync-points. The first scenario is a four-hop Bluetooth scatternet. Bluetooth networks are made up of piconets, i.e., star networks with one master and up to seven slaves, where only the master can send broadcast messages. Piconets form scatternets using nodes playing both master and slave roles. Time-stampers are nodes that must be slaves in two different piconets.

In the case of ZigBee, we implemented a mesh network similar to that of Fig. 1 but having four-hop depth. Synchronization errors for the Bluetooth and ZigBee networks are shown in Figs. 3 and 4, respectively.

As in the case of one hop, synchronization error depends chiefly on the number of points used to perform regression and the alignment error in the establishment of sync-points. We can see how as the number of hops increases, synchronization error becomes sensitive to the number of sync-points used. Again, if the application needs higher synchronization accuracy than the alignment error between nodes, it is possible to reduce error by increasing the number of sync-points to perform regression.

We find no point in comparing precision among methods because it mainly depends on the accuracy estimating sync-points and the number of pairs used. That is to say, architecture (communication transceiver, protocol, memory available, etc.) will be much more relevant than the synchronization protocol used. On the other hand, the amount of messages needed will have a big influence on the applicability of the method. This is just one of the strongest points of MBS; it drastically reduces the number of messages compared to other receiver-to-receiver protocols [4, 7]. Indeed, it is on the same order of magnitude as FTSP (the most efficient sender-to-receiver method) [5].

## Conclusions

Local clocks of nodes in wireless sensor networks have different offset and accuracy. Synchronization is mainly achieved by collecting sync-points (pairs of timestamps collected at the same time in the reference node and in the node that wants to be synchronized) and performing linear regression to compensate for differences among nodes' clocks.

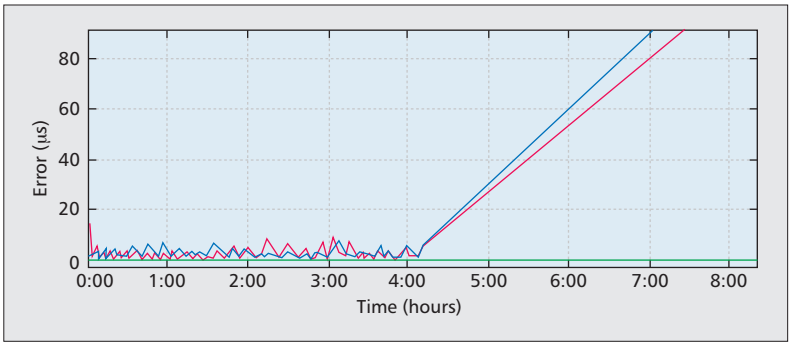



**Figure 2.** *Illustration of synchronization error vs. time with sync messages every 30 s and 20 pairs used to perform regression. Synchronization process stops after 500 sync messages.*

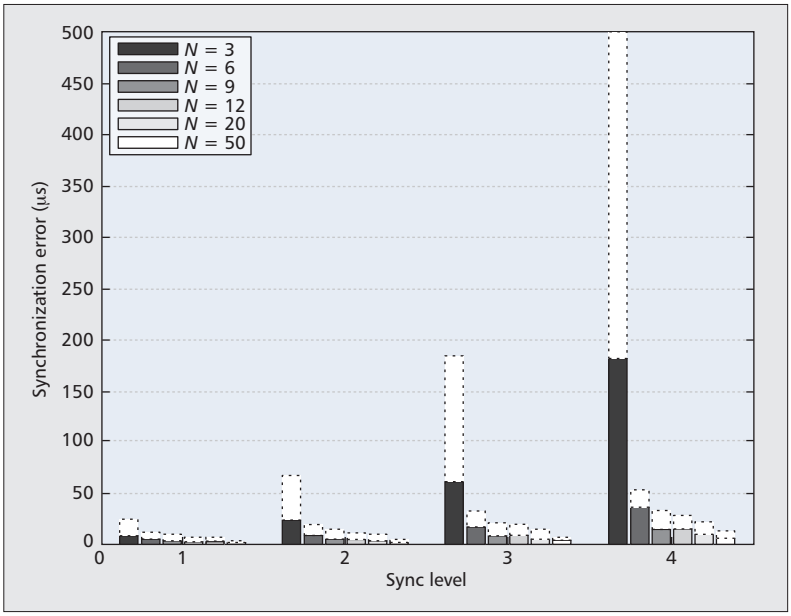



**Figure 3.** *Average synchronization error for Bluetooth network with 4 hops and sync messages every 30 s; here* N *is the number of points used to perform regression, and the dashed bars represent maximum error with 95 percent probability.*

Medium access time is a non-deterministic error that hinders accurate timestamping of transmission instants when working at the highest layers of protocol stacks. In these cases, several techniques based on a receiver-to-receiver scheme have to be used. Nevertheless, these methods set a network time shared by all the nodes (including the broadcast senders) at the expense of a high network load. This drawback may be inadmissible for large networks. In this article we have presented MBS, a multihop broadcast synchronization protocol that is able to efficiently set a common global time. The key issue of the technique is that each reference broadcast informs about the timestamp of the previous one. This way, message exchanging is minimized, similar to other sender-to-receiver methods.

We have implemented the MBS protocol in two different architectures (Bluetooth and Zig-

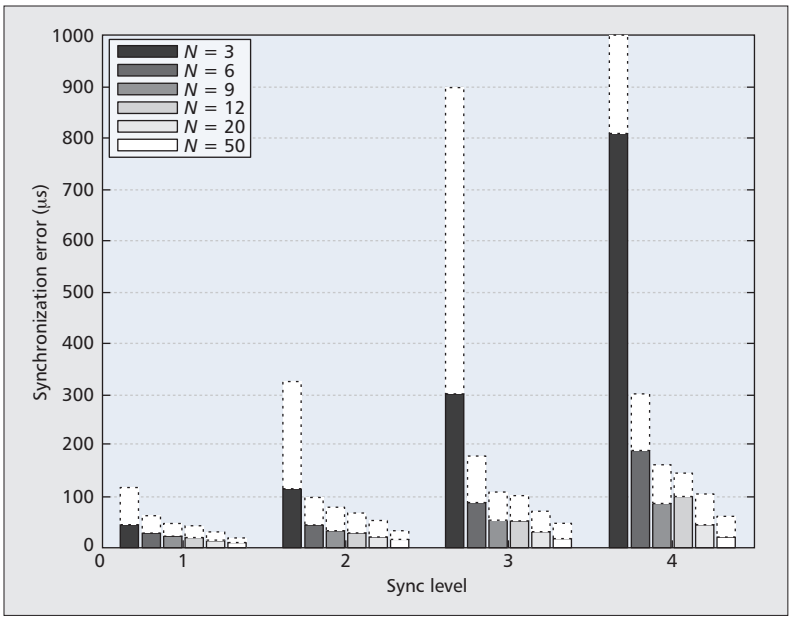



**Figure 4.** *Average synchronization error for Zigbee network with 4 hops and sync messages every 30 s; here* N *is the number of points used to perform regression, and the dashed bars represent maximum error with 95 percent probability.*

Bee), evaluating its performance. Through exhaustive experimentation we have identified the factors that influence synchronization accuracy the most. We have analyzed hardware (transceivers, crystals) and firmware (sync-message rate, number of data points to perform regression) issues to provide the application designer with guidelines that could help in deciding on the best suited technology for each application, and we have characterized synchronization behavior in several scenarios.

These results allow the application designer to decide on the hardware architecture and protocol scheme best suited to achieve the required synchronization accuracy.

## Acknowledgments

This work was supported in part by the Spanish MCYT under AmbienNet project (TIN2006-15617-C03) and by the European Commission under the MonAMI project.

## References

[1] I. F. Akyildiz *et al.*, "A Survey on Sensor Networks," *IEEE Commun. Mag.*, vol. 40, no. 8, 2002, pp. 102–14.
[2] F. Sivrikaya and B. Yener, "Time Synchronization in Sensor Networks: A Survey," *IEEE Network*, vol. 18, no. 4, 2004, pp. 45–50.
[3] A. Marco *et al.*, "Multi-Hop Synchronization at the Application Layer of Wireless and Satellite Networks," *Proc. IEEE GLOBECOM '08*, New Orleans, LA, 2008, pp. 1–5.
[4] J. Elson, L. Girod, and D. Estrin, "Fine-Grained Time Synchronization using Reference Broadcasts," *Proc. 5th Symp. Op. Sys. Design Implementation*, Boston, MA, 2002, pp. 147–63.
[5] M. Maróti *et al.*, "The Flooding Time Synchronization Protocol," Proc. 2nd ACM SenSys '04, 2004, pp. 39–49.
[6] J. Elson and D. Estrin, "Time Synchronization for Wireless Sensor Networks," *Proc. 15th Int'l. Parallel & Distrib. Process. Symp.*, San Francisco, CA, 2001, pp. 1965–70.
[7] S. PalChaudhuri, A. Saha, and D. B. Johnson, "Adaptive Clock Synchronization in Sensor Networks," *Proc. 3rd IEEE IPSN*, Berkeley, CA, 2004, pp. 340–48.
[8] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-Sync Protocol for Sensor Networks," *ACM SenSys*, Los Angeles, CA, 2003, pp. 138–49.
[9] D. Cox, A. Milenkovic, and E. Jovanov, "Time Synchronization for ZigBee Networks," *Proc. 37th South-Eastern Symp. Sys. Theory*, Tuskegee, AL, 2005, pp. 135–38.
[10] R. Casas *et al.*, "Synchronization in Wireless Sensor Networks using Bluetooth," *3rd Int'l. Wksp. Intelligent Solutions in Embedded Sys.*, 2005, pp. 79–88.
[11] J. van Greunen and J. Rabaey, "Lightweight Time Synchronization for Sensor Networks," *Proc. 2nd ACM Int'l. Conf. Wireless Sensor Net. Apps.*, San Diego, CA, 2003, pp. 11–19.