15

Time Synchronization in Wireless Sensor Networks

Jonggoo Bae and Bongkyo Moon
Dongguk University-Seoul
South Korea

1. Introduction

Recently small smart devices start to be embedded into the various environments in order to monitor the events occurred in the areas such as homes, plantations, oceans, rivers, streets, and highways. These tiny and low power devices which enable sensing and communication tasks have made sensor networks emerged. In wireless sensor networks (WSNs), especially, wireless devices get together and spontaneously form a network without any infrastructure. Due to the absence of infrastructure such as router in traditional network, nodes in a sensor network have to cooperate for communication by forwarding each other's packets from a source to its destination. Thus this yields a multi-hop communication environment.

Meanwhile, the knowledge of time between the sensor nodes is essential that detect the events such as target tracking, speed estimating, and ocean current monitoring. Hence, the sensed data often loses valuable context without accurate time information. With time synchronization, voice and video data from the different sensor nodes can be fused and displayed in a meaningful way at the sink. Time synchronization is a critical middleware service required for consistent distributed sensing and control in large-scale distributed systems such as sensor networks. That is, time synchronization in a WSN aims at providing a common time scale for local clocks of nodes in the network. Moreover, common services in WSNs, such as coordination, communication, security, power management and distributed logging also depend on the global time scale.

The most widely adapted time synchronization protocol in the internet domain is the Network Time Protocol (NTP) devised by Mills (Mills, 1991). Nodes could also be equipped with a global positioning system (GPS) to synchronize them (Hofmann-Wellenhof et al. 1997; Mannermaa et al. 1999). It is used to provide network-wide agreement among a large group of nodes in the Internet. 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. Consequently, it may be useful to use NTP to discipline sensor nodes, but 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. Therefore, without further adaptation, NTP is suitable only for WSN applications with low precision demands.

Time synchronization is a key service for many applications and operating systems in distributed computing environments. WSNs are large-scale distributed systems, but traditional distributed algorithms cannot be considered for problems due to their unique characteristics, especially the severe resource constraints. In this chapter, the mechanisms to synchronize the local clocks of the nodes in WSN have been extensively investigated.

2. Backgrounds and Related Works

A landmark study in computer clock synchronization is Lamport's work that elucidates the importance of *virtual clocks* in systems where causality is more important than absolute time (Lamport, 1978). Though Lamport's work focused on giving events a total order rather than quantifying the time difference between them, it has emerged as an important influence in sensor networks. Many sensor applications require only relative time, for example, timing the propagation delay of sound (Girod & Estrin, 2001), and thus absolute time may not be needed. Mills' NTP (Mills, 1991) stands out by virtue of its scalability, self-configuration in large multi-hop networks, robustness to failures and sabotage, and ubiquitous deployment. NTP allows construction of a hierarchy of time servers, multiply rooted at canonical sources of external time.

Post-facto synchronization was a pioneering work by Elson and Estrin (Elson & Estrin, 2001). In this approach, unlike in traditional synchronization schemes such as NTP, each node's clock is normally unsynchronized with the rest of the network; a beacon node periodically broadcasts beacon messages to the sensor nodes in its wireless range. When an event is detected, each node records the time of the event (timestamp with its own local clock). After the event (hence the name), upon receiving the reference beacon message, nodes use it as time reference and adjust their event timestamps with respect to that reference. This synchronization scheme has led afterwards to their RBS (Reference Broadcast Synchronization) protocol.

Elson et al. propose a scheme called Reference-Broadcast Synchronization (RBS), in which a node sends reference broadcast beacons to its neighbors using physical layer broadcasts (Elson et al., 2002). RBS gets around the non-determinism of packet send time, access time, and propagation time, while depending only on the packet receive time. Since the packet receive time is the same for all receivers, this reference broadcast packet can be used to synchronize a set of receivers with one another. This scheme can also be extended to a multi-hop scenario. However, the impact of the translation errors and delays on the multi-hop synchronization, which can be provided by translating the time between different broadcast domains, still needs to be studied. In addition, they do not consider global synchronization over the entire network.

A more recently developed *Time-Sync protocol for Sensor Networks* (TPSN) (Ganeriwal et al., 2003) is based on similar methodology as the NTP, where the sensor nodes are organized into multiple levels and synchronized to the root node of the hierarchy. Unlike the Internet, the root node and nodes at different levels responsible for synchronization may fail often, which may cause synchronization problems. In addition, mobile nodes may disrupt the predefined level-by-level synchronization procedure. On typical WSN platforms using the TPSN protocol, such as the Mica2 mote, it is possible to access directly to the MAC layer, and message time-stamping can be performed during message transmission and reception. This immediately eliminates the same three main sources of uncertainties as in RBS. With a

two-way handshake of synchronization messages, the TPSN protocol eliminates the unknown propagation time as well. Although the propagation time has been eliminated, the encoding and decoding times are not because they might not be the same on the sender and receiver side. It is important to point out that both the RBS and TPSN protocols suffer from the two largest sources of uncertainty of MAC layer time-stamping: the jitter of interrupt handling and decoding time.

On the other hand, the flooding time synchronization protocol (FTSP) effectively reduces all sources of time stamping errors except for the propagation time. The FTSP (Maroti et al. 2004) was designed for a sniper localization application requiring very high precision (Simon et al. 2004). FTSP achieves the required accuracy by utilizing a customized MAC-layer time stamping and by using calibration to eliminate unknown delays. FTSP is robust to network failures, as it uses flooding both for pair-wise and global synchronization. Linear regression from multiple timestamps is used to estimate the clock drift and offset. The main drawback of FTSP is that it requires calibration on the hardware actually used in the deployment (thus is not a software solution purely independent of the hardware). FTSP also requires intimate access to the MAC layer for multiple timestamps. However, if well-calibrated, the FTSP's precision is impressive (less than $2\mu s$).

Su and Akyildiz proposed the time-diffusion synchronization protocol (TDP) for network-wide time synchronization (Su & Akyildiz, 2005). The main idea of TDP is to start from a master node, adjust the clocks of its neighbors, and diffuse this clock adjustment to other nodes. TDP maintains global time synchronization within an adjustable bound based on the application requirements. One of the benefits of TDP is that the performance of voice and video applications can be improved when multiple sources send data back to the sink through flooding or *directed diffusion* (Intanagonwiwat et al. 2003). It achieves global synchronization by multi-hop flooding: The base station initiates the protocol by sending a special timing message to the entire network. Some of the nodes, upon receiving the message, become masters by using a leader election procedure. The master nodes start the time-diffusion procedure involving electing diffused leaders, multi-hop flooding, and iterative weighted averaging of timings from different master nodes. TDP handles node mobility and failures by using a peer evaluation procedure.

3. Time Synchronization

3.1 Clocks and Synchronization

3.1.1 Sensor Node Clock

Every sensor node maintains its own clock and this is the only notion of time that a node has. The clock is an ensemble of hardware and software components; it is essentially a timer that counts the oscillations of a quartz crystal running at a particular frequency. Computing devices are mostly equipped with a hardware oscillator assisted computer clock, which implements an approximation $C(t)$ of real-time t. Let us represent the clock for node A by $C_A(t)$. The difference in the clocks of two sensor nodes (i.e., A and B) is referred as the offset error between them. There are three reasons for the nodes to be representing different times in their respective clocks (Ganeriwal et al. 2008): 1) The nodes might have been started at different times, 2) the quartz crystals at each of these nodes might be running at slightly different frequencies, causing the clock values to gradually diverge from each other (termed

as the skew error), or 3) the frequency of the clocks can change differently over time because of aging or ambient conditions such as temperature (termed as the drift error). These errors can be summarized as follows:

$$\text{Offset: } \delta = C_A(t) - C_B(t) \tag{1}$$

$$\text{Skew: } \eta = \frac{\partial C_A(t)}{\partial t} - \frac{\partial C_B(t)}{\partial t} \tag{2}$$

$$\text{Drift: } \lambda = \frac{\partial^2 C_A(t)}{\partial t^2} - \frac{\partial^2 C_B(t)}{\partial t^2} \tag{3}$$

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 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 (Sichitiu & Veerarittiphan, 2003). Then, for some node *i* in the network, we can approximate its local clock as:

$$C_i(t) = a_i t + b_i \tag{4}$$

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 (4), we can compare the local clocks of two nodes in a network, say node i and node j as:

$$C_i(t) = a_{ij} \cdot C_j(t) + b_{ij} \tag{5}$$

We call a_{ij} the *relative drift*, and b_{ij} the *relative offset* between the clocks of node i and node j. If two clocks are perfectly synchronized, then their relative drift is i (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. Also, the "offset" may equivalently be mentioned as "phase offset". Fig. 1 shows the relationship between relative drift and offset.

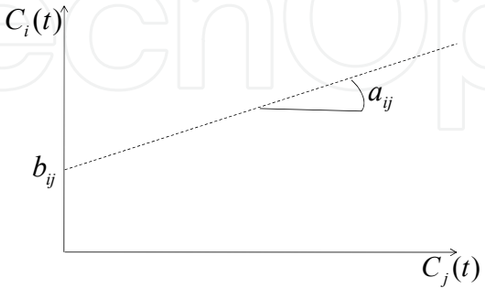


Fig. 1. The relation between relative drift and offset

Although each sensor node is equipped with a hardware clock, these hardware clocks can usually not be used directly, as they suffer from severe drift. No matter how well the

hardware clocks will be calibrated at deployment, the clocks will ultimately exhibit a large skew. 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. To allow for an accurate common time, nodes need to exchange messages from time to time, constantly adjusting their clock values. Furthermore, nodes can convert the current hardware clock reading into a logical clock value and vice versa (Sommer & Wattenhofer, 2009).

- Hardware Clock
Each sensor node *i* is equipped with a hardware clock $H_i(\cdot)$. The clock value at time *t* is defined as

$$H_i(t) = \int_{t_0}^{t} h_i(\tau) d\tau + \Phi_i(t_0)$$

where $h_i(\tau)$ is the hardware clock rate at time τ and $\Phi_i(t_0)$ is the hardware clock offset at time t_0. It is assumed that hardware clocks have bounded drift, i.e., there exists a constant $0 \le \rho <$ 1 such that $1 - \rho \le h(t) \le 1 + \rho$ for all times *t*. This implies that the hardware clock never stops and always makes progress with at least a rate of $1 - \rho$. This is a reasonable assumption since common sensor nodes are equipped with external crystal oscillators which are used as clock source for a counter register of the microcontroller. These oscillators exhibit drift which is only gradually changing depending on the environmental conditions such as ambient temperature or battery voltage and on oscillator aging. This allows assuming the oscillator drift to be relatively constant over short time periods. Crystal oscillators used in sensor nodes normally exhibit a drift between 30 and 100 ppm (Sommer & Wattenhofer, 2009).

- Logical Clock
Since other hardware components may depend on a continuously running hardware clock, its value should not be adjusted manually. Instead, a logical clock value $L_i(\cdot)$ is computed as a function of the current hardware clock. The logical clock value $L_i(t)$ represents the synchronized time of node *i*. It is calculated as follows:

$$L_i(t) = \int_{t_0}^{t} h_i(\tau) \cdot l_i(\tau) d\tau + \theta_i(t_0)$$

where $l_i(\tau)$ is the *relative logical clock rate* and $\theta_i(t_0)$ is the clock offset between the hardware clock and the logical clock at the reference time t_0. The logical clock is maintained as a software function and is only calculated on request based on a given hardware clock reading (Sommer & Wattenhofer, 2009).

3.1.2 Definition of Clock Synchronization

The synchronization problem on a network of *n* devices corresponds to the problem of equalizing the computer clocks of the 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 the nodes that are spatially close. Equalizing just the instantaneous values of clocks by correcting the offsets 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 in order to keep the clocks synchronized over a time period (Sivrikaya & Yener, 2004) .

The above definition of synchronization actually defines the strictest form of synchronization, where one seeks perfect matching of time on different clocks, but this definition can be relaxed to different degrees according to the needs of an application. In general, the synchronization problem can be classified into three basic types (Ganeriwal et al. 2003). First 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 $E1$ has occurred before or after another event $E2$, i.e. just to compare the local clocks for order rather than having them synchronized. The algorithm proposed in (Romer, 2003) is an example to this type of synchronization. Second type of synchronization algorithms targets 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 (Elson et al. 2002; Sichitiu & Veerarittiphan, 2003). The third 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 (Ganeriwal et al. 2003) conforms to this model, but the use of "always on" mode is not mandatory in the scheme.

3.2 Design Factors for Time Synchronization

Some of the factors influencing time synchronization in wireless sensor networks are temperature, phase noise, frequency noise, asymmetric delays, and clock glitches (Su & Akyildiz, 2005).

- Temperature: Since sensor nodes are deployed in various places, the temperature variations throughout the day may cause the clock to speed up or slow down. For a typical sensor node, the clock drifts few *parts per million (ppm)* during the day (Mills, 1998). For low-end sensor nodes, the drifting may be even worse.
- Phase noise: Some of the causes of phase noise are access fluctuations at the hardware interface, response variation of the operating system to interrupts, and jitter in the network delay. The jitter in the network delay may be due to medium access and queueing delays.
- Frequency noise: The frequency noise is due to the unstability of the clock crystal. A low-end crystal may experience large frequency fluctuation, because the frequency spectrum of the crystal has large sidebands on adjacent frequencies.
- Asymmetric delay: Since sensor nodes communicate with each other through the wireless medium, the delay of the path from one node to another may be different than the return path. As a result, an asymmetric delay may cause an offset to the clock that cannot be detected by a variance type method (Levine, 1999). If the asymmetric delay is static, the time offset between any two nodes is also static. The

asymmetric delay is bounded by one-half the round trip time between the two nodes (Levine, 1999).

• Clock glitches: Clock glitches are sudden jumps in time. This may be caused by hardware or software anomalies such as frequency and time steps. Besides dealing with these factors, a time synchronization protocol for sensor networks should be *automatically self-configured* and be *sensitive to energy requirement*.

3.3 Synchronization Problems in WSNs

Network time protocol (NTP) (Mills 1991) has been widely used in the Internet for decades. The NTP clients synchronize their clocks to the NTP time servers with accuracy in the order of milliseconds by statistical analysis of the round-trip time. The time servers are synchronized by external time sources, typically using GPS. The NTP has been widely deployed and proved to be effective, secure and robust in the internet. However, traditional synchronization schemes and GPS-equipped systems are not suitable for use in WSNs due to the specific requirements of those networks (Yoon et al. 2007):

• Precision: WSNs may require much higher precision than traditional networks depending on the deployed applications. For example, a precision of a few milliseconds is satisfactory for NTP in the Internet, while microsecond precision may be required in order to significantly improve the performance of the WSN beam-forming application.

• Cost: Nodes in WSNs typically have limited batteries, computational resources, and storage capacity. However, most of the protocols designed for wired environments need to exchange many messages and also store them for statistical processing.

The problem in a modern sensor network scenario is that nodes can only communicate locally to their neighbors. The localized communication makes the problem much harder in that: 1) a valid consensus has to be computed locally and 2) the local consensus must be conveyed to other parts of the network; this is even harder because the relay nodes may be faulty or malicious. In order to provide network-wide time synchronization, the time differences among the sensor nodes must be minimized before protocols requiring time-stamps (e.g., security applications, flow control protocols, target tracking, voice fusion, video fusion, and environmental data fusion) are realizable. In addition, the time synchronization protocol must be robust to node failures as well as energy consumption in the network .

Typically the synchronization problems in wireless sensor networks need to be addressed for the following reasons (Sivrikaya & Yener, 2004). First, sensor nodes need to coordinate their operations and collaborate each other in order to achieve a complex sensing task. That is, data fusion is made through aggregating data collected from different nodes for a meaningful result. Second, power saving function requires synchronization for increasing network lifetime. For power saving, sensors may *sleep* by turning off their sensors and/or transceivers at appropriate times, and wake up at coordinated times. However, the radio receiver of a sensor node is not turned off in the case that there are some data directed to it. This requires a precise timing between sensor nodes. Third, scheduling algorithms in WSNs are used to share the transmission medium in the time domain to eliminate transmission

collisions and conserve energy. However, non-determinism in transmission time caused by the Media Access Channel (MAC) layer of the radio stack can introduce several hundreds of milliseconds delay at each hop. Thus, synchronization is an essential part of transmission scheduling.

3.4 Uncertainties and Errors in Time Synchronization

Time synchronization schemes rely on some sort of message exchange between nodes in WSN. Non-determinism in the network dynamics such as propagation time or physical channel access time makes the synchronization task a big challenge in many systems. Note that in short distance multi-hop broadcast, the data processing time and its variation contribute the most to time fluctuations and differences in the path delays. Also, the time difference between two sensor nodes may become large over time due to the wandering effect of the local clocks. Latency estimates are actually confounded by random events that lead to asymmetric round-trip message delivery delays; this delay prevents the receiver from exactly comparing the local clocks of the two nodes and accurately synchronizing to the sender node. To better understand the source of these errors, it is useful to decompose the source of a message's latency. Kopetz and Ochsenreiter (Kopetz & Ochsenreiter, 1987) introduced firstly four distinct components for analyzing the sources of the message delivery delays and later extended in (Ganeriwal et al. 2003).

- *Send Time:* The time spent at the sender to construct the message. This includes kernel protocol processing and variable delays introduced by the operating system (e.g., context switches and system call overhead occurred by the synchronization application), and the time to transfer the message from the host to its network interface for transmission.
- Access Time: Each packet faces some delay at the MAC (Medium Access Control) layer before actual transmission. This delay is specific to the MAC protocol in use, 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. When the sender and receiver share access to the same physical media (e.g., neighbors in an ad-hoc wireless network, or on a LAN), this delay is very small as it is simply the physical propagation time of the message through the media.
- Receive Time: This is the processing time required for the receiver's network interface to receive the message from the channel and notify the host of its arrival. This is typically the time required for the network interface to generate a message reception signal. If the arrival time is time-stamped at a enough low level in the host's operating system kernel, this delay does not include the overhead of system calls, context switches, or even the message transfer from the network interface to the host.
- Transmission Time: The time it takes for the sender to transmit the message. This time is in the order of tens of milliseconds depending on the length of the message and the speed of the radio.

• Reception Time: The time it takes for the receiver to receive the message. It is the same as the transmission time. The transmission and reception times overlap in WSN as pictured in Fig. 2.

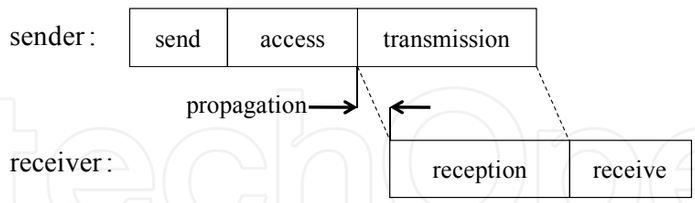


Fig. 2. Decomposition of the message delivery delay over a wireless link (Maroti, et al. 2004)

• Interrupt Handling Time: The delay between the radio chip raising and the microcontroller responding to an interrupt. This time is mostly less than a few microsecond (waiting for the microcontroller to finish the currently executed instruction), however, when interrupts are disabled this delay can grow large.

• Encoding Time: The time it takes for the radio chip to encode and transform a part of the message to electromagnetic waves starting from the point when it raised an interrupt indicating the reception of the idealized point from the microcontroller. This time is deterministic and is in the order of a hundred microseconds.

• Decoding Time: The time it takes for the radio chip on the receiver side to transform and decode the message from electromagnetic waves to binary data. It ends when the radio chip raises an interrupt indicating the reception of the idealized point. This time is mostly deterministic and is in the order of hundred microseconds. However, signal strength fluctuations and bit synchronization errors can introduce jitter.

• Byte Alignment Time: The delay incurred because of the different byte alignment of the sender and receiver. This time is deterministic and can be computed on the receiver side from the bit offset and the speed of the radio.

Fig. 3 summarizes the decomposition of delivery delay of the idealized point of the message as it traverses over a wireless channel. Each line represents the time line of the layer as measured by an ideal clock. The dots represent the time instance when the idealized point of the message crosses the layers. The triangles on the first and last line represent the time when the CPU makes the time-stamps. Depending on the specific hardware the time stamp is usually recorded by the microcontroller when it handles the radio chip interrupts both on the sender and receiver sides. Alternatively, capture registers provided by some hardware can be employed to eliminate the interrupt handling time (Maroti, et al. 2004).

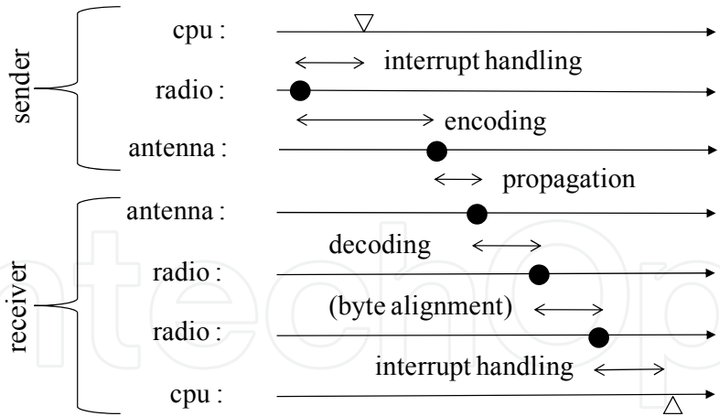


Fig. 3. The timing of the transmission of an idealized point in the software (cpu), hardware (radio chip) and physical (antenna) layers of the sender and the receiver (Maroti, et al. 2004)

Table 1 summarizes the magnitudes and distribution of the various delays in message transmissions on the Mica2 platform. The block codes are used, and the idealized point of the message can also be assumed to be at a block boundary (Maroti, et al. 2004).

Time	**Magnitude**	**Distribution**
Send & Receive	0 – 100 ms	nondeterministic, depends on the processor load
Access	10 – 500 ms	nondeterministic, depends on the channel contention
Transmission & Reception	10 – 20 ms	deterministic, depends on message length
Propagation	< 1μs for distances up to 300 meters	deterministic, depends on the distance between sender and receiver
Interrupt Handling	< 5μs in most cases, but can be as high as 30μs	nondeterministic, depends on interrupts being disabled
Encoding plus Decoding	100 – 200 μs < 2 μs variance	deterministic, depends on radio chipset and settings
Byte Alignment	0 – 400μs	deterministic, can be calculated

Table 1. The sources of delays in message transmissions (Maroti, et al. 2004)

3.5 Metrics for Evaluating Time Synchronization Schemes

The requirements for the synchronization problem can be regarded as the metrics for evaluating synchronization schemes on wireless sensor networks. Combining with the criteria that sensor nodes have to be energy efficient, low-cost, and small in a multi-hop environment, this requirement becomes a challenging problem to solve. However, a single synchronization scheme may not satisfy them all together since there are actually tradeoffs between the requirements of an efficient solution (Sivrikaya & Yener, 2004).

• 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 [1]secs.

• 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 cannot 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. Time Synchronization Methods

Time synchronization has been a seminal topic in distributed systems (Dolev et al. 1984; Halpern et al. 1984; Lundelius et al. 1984; Lamport et al. 1985), but designing clock synchronization algorithms in the context of a sensor network is challenging for several reasons. First, traditional distributed systems assume that all the nodes in a network can communicate directly with each other. A sensor network, however, is subject to spatial

constraints. Nodes only communicate directly with their neighbors. Communication between two remote nodes is accomplished by message relay using intermediate nodes. Second, nodes in a sensor network generally rely on less information about the system than traditional distributed systems, where nodes have access to the clock values of all the other members of the system, including the faulty nodes. Third, a sensor node has only limited processing capability. The computation intensive signature algorithms, such as RSA, are not suitable for sensor networks. Instead, some light-weight algorithms (such as using a one-way key chain or a key management scheme) are more suitable. The spatial constraints, the communication cost and delay, and the diminished computational capability are key reasons why localized algorithms that involve lightweight computations are preferred for sensor networks.

4.1 RBS(Reference Broadcast Synchronization)

The main advantage of RBS is that it eliminates transmitter-side non-determinism. The disadvantage of the approach is that additional message exchange is necessary to communicate the local time-stamps between the nodes. Eventually the RBS approach completely eliminates the send and access times, and with minimal OS modifications it is also possible to remove the receive time uncertainty. This leaves the mostly deterministic propagation and reception time in wireless networks as the sole source of error. The main strength of RBS is its broad applicability to commodity hardware and existing software in sensor networks as it does not need access to the low levels of the operating system (Elson et al. 2002).

The novel idea in RBS scheme is to use a third party for synchronization instead of synchronizing the sender with a receiver. This scheme synchronizes a set of receivers with one another. Although its application in sensor networks is novel, the idea of *receiver-receiver synchronization* was previously proposed for synchronization in broadcast environments. 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 (Sivrikaya & Yener, 2004).

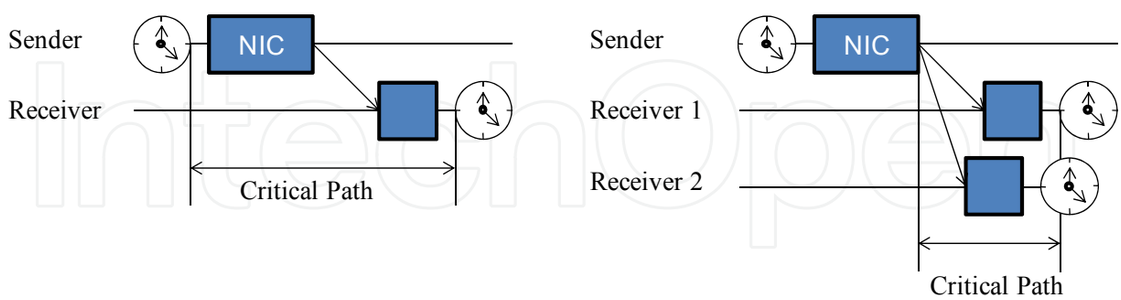


Fig. 4. Critical path analysis between traditional time synchronization protocol (*left*) and RBS (*right*) (Elson et al. 2002)

By removing the sender's non-determinism from the critical path (Fig. 4), RBS scheme achieves much better precision compared to traditional synchronization methods that use two-way message exchanges between synchronizing nodes. As the sender's non-determinism has no effect on RBS precision, the only sources of error can be the non-

determinism in propagation time and receive time. In this scheme, 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 the only receive time errors are handled when the accuracy of RBS model is analyzed (Elson et al. 2002; Sivrikaya & Yener, 2004) .
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.

4.2 TPSN (Timing-Sync Protocol for Sensor Network)

The TPSN algorithm first creates a spanning tree of the network and then performs pair-wise synchronization along the edges. Each node gets synchronized by exchanging two synchronization messages with its reference node one level higher in the hierarchy. The TPSN achieves two times better performance than RBS by time-stamping the radio messages in the Medium Access Control (MAC) layer of the radio stack (Ganeriwal et al., 2003) and by relying on a two-way message exchange. The shortcoming of TPSN is that it does not estimate the clock drift of nodes, which limits its accuracy, and does not handle dynamic topology changes.
The first step of the algorithm is to create a hierarchical topology in the network. Every node is assigned a level in this hierarchical structure, and a node belonging to level *i* can communicate with at least one node belonging to level *i-1*. Only one node is assigned to level *0*, which is called the "root node". This stage of the algorithm is called as the "level discovery phase". Once the hierarchical structure has been established, the root node initiates the second stage of the algorithm, which is called the "synchronization phase". In this phase, a node belonging to level *i* synchronize to a node belonging to level *i-1*. Eventually every node is synchronized to the root node and network-wide time synchronization is achieved (Ganeriwal et al., 2003).

4.2.1 Level Discovery Phase

This phase of the algorithm occurs at the onset, when the network is deployed. The root node is assigned a level *0* and it initiates this phase by broadcasting a *level_discovery* packet. The *level_discovery* packet contains the identity and the level of the sender. The immediate neighbors of the root node receive this packet and assign themselves a level, one greater than the level they have received i.e., level *1*. After establishing their own level, they broadcast a new *level_discovery* packet containing their own level. This process is continued and eventually every node in the network is assigned a level. On being assigned a level, a node neglects any such future packets. This makes sure that no flooding congestion takes place in this phase. Thus a hierarchical structure is created with only one node, root node, at level *0*. A node might not receive any *level_discovery* packets owing to MAC layer collisions (Ganeriwal et al., 2003).

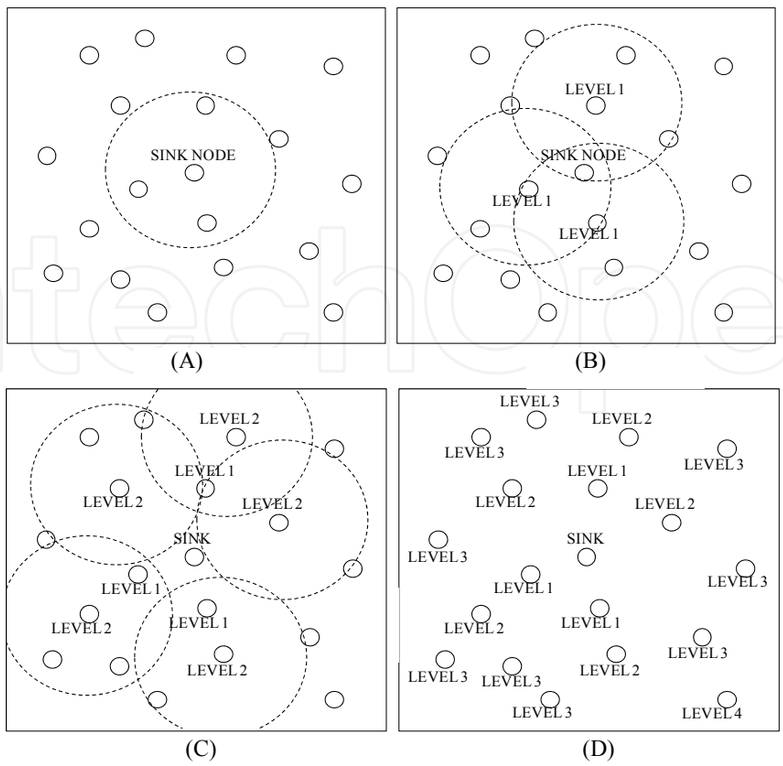


Fig. 5. The Process of level discovery phase for hierarchical topology organization in TPSN

4.2.2 Synchronization Phase

In this phase, pair wise synchronization is performed along the edges of the hierarchical structure established in the earlier phase. The classical approach of sender-receiver synchronization (Mills, 1991) is used for doing this handshake between a pair of nodes. Fig. 6 shows this message-exchange between nodes '*A*' and '*B*'. Here, *T1, T4* represent the time measured by local clock of '*A*'. Similarly *T2, T3* represent the time measured by local clock of '*B*'. At time *T1*, '*A*' sends a *synchronization_pulse* packet to '*B*'. The *synchronization_pulse packet* contains the level number of '*A*' and the value of *T1*. Node *B* receives this packet at *T2*, where *T2* is equal to *T1* + D + *d*. Here D and *d* represents the clock drift between the two nodes and propagation delay respectively. At time *T3*, '*B*' sends back an *acknowledgement* packet to '*A*'. The *acknowledgement* packet contains the level number of '*B*' and the values of *T1, T2* and *T3*. Node *A* receives the packet at *T4*. Assuming that the clock drift and the propagation delay do not change in this small span of time, '*A*' can calculate the clock drift and propagation delay as (Ganeriwal et al., 2003) :

$$\Delta = \left(\frac{T2 - T1) - (T4 - T3)}{2}\right) \; ; \; d = \left(\frac{T2 - T1) + (T4 - T3)}{2}\right) \tag{6}$$

Knowing the drift, node *A* can correct its clock accordingly, so that it synchronizes to node *B*. This is a sender initiated approach, where the sender synchronizes its clock to that of the receiver.

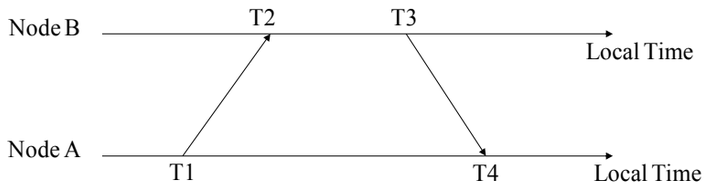


Fig. 6. Two way message exchange between a pair of nodes (Ganeriwal et al., 2003)

This message exchange at the network level begins with the root node initiating the phase by broadcasting a *time_sync* packet. On receiving this packet, nodes belonging to level *1* wait for some random time before they initiate the two-way message exchange with the root node. This randomization is to avoid the contention in medium access. On receiving back an acknowledgment, these nodes adjust their clock to the root node. The nodes belonging to level *2* will overhear this message exchange. This is based on the fact that every node in level *2* has at least one node of level *1* in its neighbor set. On hearing this message, nodes in level *2* back off for some random time, after which they initiate the message exchange with nodes in level *1* (Ganeriwal et al., 2003).
This randomization is to ensure that nodes in level *2* start the synchronization phase after nodes in level *1* have been synchronized. Note that a node sends back an *acknowledgement* to a *synchronization_pulse*, provided that it has synchronized itself. This ensures that no multiple levels of synchronization are formed in the network. This process is carried out throughout the network and eventually every node is synchronized to the root node. In a sensor network, packet collisions can take place quite often. To handle such scenario a node waiting for an acknowledgement, timeouts after some random time and retransmits the *synchronization_pulse*. This process is continued until a successful two-way message exchange has been done (Ganeriwal et al., 2003).

4.3 FTSP(Flooding Time Synchronization Protocol)

The goal of the FTSP is to achieve a network wide synchronization of the local clocks of the participating nodes. In this protocol, each node has a local clock exhibiting the typical timing errors of crystals and can communicate over an unreliable but error corrected wireless link to its neighbors. The FTSP synchronizes the time of a sender to possibly multiple receivers utilizing a single radio message time-stamped at both the sender and the receiver sides. MAC layer time-stamping can eliminate many of the errors, as observed in many previous protocols (Ganeriwal et al., 2003; Woo & Culler, 2001). However, accurate time-synchronization at discrete points in time is a partial solution only. Compensation for the clock drift of the nodes is inevitable to achieve high precision between synchronization points and to keep the communication overhead low. Linear regression is used in FTSP to compensate for clock drift as suggested in (Elson et al., 2002).
Typical WSN operate in areas larger than the broadcast range of a single node; therefore, the FTSP provides multi-hop synchronization. The root of the network, a dynamically elected single node, maintains the global time and all other nodes synchronize their clocks to that of the root. The nodes form an ad-hoc structure to transfer the global time from the root to all the nodes, as opposed to a fixed spanning-tree based approach proposed in (Ganeriwal et al.,

2003). This saves the initial phase of establishing the tree and is more robust against node and link failures and dynamic topology changes.

4.3.1 Time-stamping

The FTSP utilizes a radio broadcast to synchronize the possibly multiple receivers to the time provided by the sender of the radio message. The broadcasted message contains the sender's time stamp which is the estimated global time at the transmission of a given byte. The receivers obtain the corresponding local time from their respective local clocks at message reception. Consequently, one broadcast message provides a *synchronization point* (a global-local time pair) to each of the receivers (Maroti et al. 2004). The difference between the global and local time of a synchronization point estimates the clock offset of the receiver. As opposed to the RBS protocol, the time stamp of the sender must be embedded in the currently transmitted message. Therefore, the time-stamping on the sender side must be performed before the bytes containing the time stamp are transmitted.

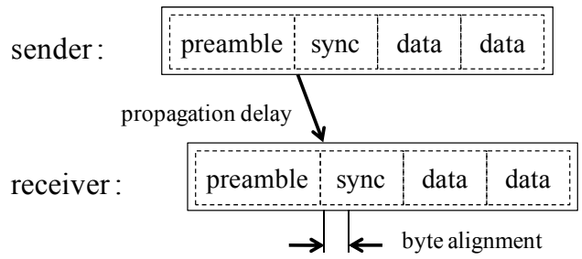


Fig. 7. Data packets transmitted over the radio channel. Solid lines represent the bytes of the buffer and the dashed lines are the bytes of packets (Maroti et al. 2004)

Message broadcast starts with the transmission of preamble bytes, followed by SYNC bytes, then with a message descriptor followed by the actual message data, and ends with CRC bytes. During the transmission of the preamble bytes the receiver radio synchronizes itself to the carrier frequency of the incoming signal. From the SYNC bytes the receiver can calculate the bit offset it needs to reassemble the message with the correct byte alignment. The message descriptor contains the target, the length of the data and other fields, such as the identifier of the application layer that needs to be notified on the receiver side. The CRC bytes are used to verify that the message was not corrupted. The message layout is summarized in Fig. 7.

The FTSP time-stamping effectively reduces the jitter of the interrupt handling and encoding/decoding times by recording multiple time stamps both on the sender and receiver sides. The time stamps are made at each byte boundary after the SYNC bytes as they are transmitted or received. First, these time stamps are normalized by subtracting an appropriate integer multiple of the nominal byte transmission time, the time it takes to transmit a byte. The jitter of interrupt handling time is mainly due to program sections disabling interrupts on the microcontroller for short amounts of time. This error is not Gaussian, but can be eliminated with high probability by taking the minimum of the normalized time stamps. The jitter of encoding and decoding time can be reduced by taking the average of these interrupt error corrected normalized time stamps. On the receiver side this final averaged time stamp must be further corrected by the byte alignment time that can be computed from the transmission speed and the bit offset (Maroti et al. 2004).

4.3.2 Clock drift management

If the local clocks had the exact same frequency and, hence, the offset of the local times were constant, a single synchronization point would be sufficient to synchronize two nodes. However, the frequency differences of the crystals used in Mica2 motes introduce drifts up to 40μs per second. This would mandate continuous re-synchronization with a period of less than one second to keep the error in the micro-second range, which is a significant overhead in terms of bandwidth and energy consumption (Maroti et al. 2004). Therefore, it is necessary to estimate the drift of the receiver clock with respect to the sender clock. The offset between the two clocks changes in a linear fashion provided the short term stability of the clocks is good. In this scheme, the stability of the 7.37 MHz Mica2 clock is verified by periodically sending a reference broadcast message that was received by two different motes. The two motes time-stamped the reference message using the FTSP time-stamping described in the previous section with their local time of arrival and reported the time-stamp (Maroti et al. 2004).

4.4 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 (Sichitiu & Veerarittiphan, 2003). The authors assume that each clock can be approximated by an oscillator with fixed frequency. As argued in previous section, two clocks, $C_1(t)$ and $C_2(t)$, can be linearly related under this assumption as:

$$C_1(t) = a_{12} \cdot C_2(t) + b_{12} \tag{7}$$

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, time stamped 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 (7), the following inequalities can be obtained:

$$t_o < a_{12} \cdot t_b + b_{12} \tag{8}$$

$$t_r > a_{12} \cdot t_b + b_{12} \tag{9}$$

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 (Sichitiu & Veerarittiphan, 2003). Each data point corresponds to two constraints on the relative drift and relative offset (equations 8, 9). The constraints imposed by data points are depicted in Fig. 8. Note that the line corresponding to equation (9) must lie between the vertical intervals created by each data point. One of the dashed lines in Fig. 8 represent the steepest possible such line, satisfying equation (7). 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:

$$\underline{a_{12}} \leq a_{12} \leq \overline{a_{12}} \tag{10}$$

$$\underline{b_{12}} \leq b_{12} \leq \overline{b_{12}} \tag{11}$$

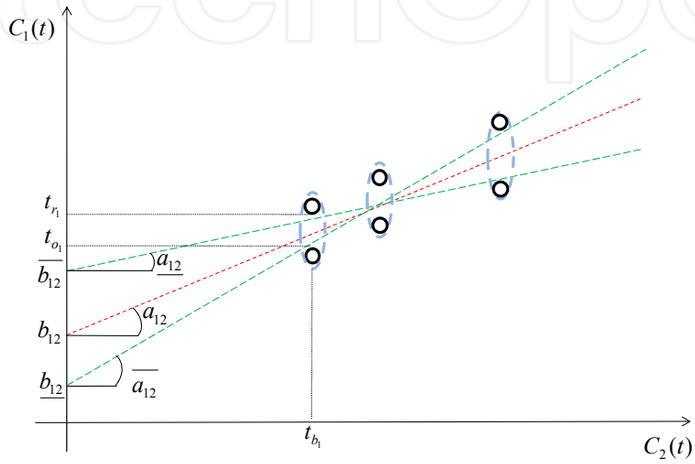


Fig. 8. The constraints imposed on a_{12} and b_{12} by data points (Sivrikaya & Yener, 2004)

The 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 (Sichitiu & Veerarittiphan, 2003; Sivrikaya & Yener, 2004).

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 Fig. 8 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 founds 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 (Sivrikaya & Yener, 2004).

5. Global Time Synchronization Algorithms

Li and Rus (Li & Rus, 2006) presented a high-level framework for global synchronization. The three methods are proposed for global synchronization in WSNs. The first two methods, all-node-based and cluster-based synchronization, use global information but are not suitable for large WSNs. In the third approach, diffusion method, each node sets its clock to the average clock time of its neighbors. The diffusion method thus converges to a global average value. A drawback of this approach is the potentially large number of messages exchanged between neighbor nodes, especially in dense networks.

5.1 All-Node-based Synchronization

This method is used on all the nodes in the system and it is most effective when the size of the sensor network is relatively small. In future sections of this paper, they describe ways to address scalability. They assume the clock cycle on each node is the same. They believe this is a reasonable assumption since most sensors are programmed with the same parameters prior to deployment. They also assume the clock tick time is much longer than the packet transmission time. Finally, they assume that the message transmission time over each link and handling time on each node are roughly the same. This time can be obtained when the network traffic is small. That is, upon its initial deployment, a sensor network allows sufficient time solely for clock synchronization. The key idea is to send a message along a loop and record the initial time and the end time of the message. Then, by using the message traveling time, they can average the time to different segments of the loop and smooth over the error of the clocks. Algorithm 1 (Li & Rus, 2006) summarizes this method.

Algorithm 1 All-Node-Based Synchronization Algorithms in a Sensor Network

1: Find a ring that passes each node at least once that need to be synchronized (suppose the ring is composed of k nodes)
2: A message is passed along the ring starting from an initiating node
3: Upon receipt of the message, each node records its current local time (t_i) and its order (i) in the ring. If the node receives messages more than once, it chooses one arbitrarily
4: After initiating node receives the message, it sends out another message informing each node on the ring the start time (t_s) and the end time (t_e) of the previous message
5: **for** each node, to adjust its local time t **do**
6: **if** $\exists\, m, m+1 \geq \frac{t_e - t_s + 1}{k} \cdot (i-1) \geq t_i \geq \frac{t_e - t_s}{k} \cdot (i-1) \geq m$ **then**
7: node n_i adjusts its time to $t - t_i + t_s + m$
8: **if** $\exists\, m, m+1 \geq \frac{t_e - t_s + 1}{k} \cdot (i-1) \geq t_i \geq \frac{t_e - t_s}{k} \cdot (i-1) \geq m-1$ **then**
9: node n_i adjusts its time to $t - t_i + t_s + m$

5.2 Cluster-based Synchronization

The synchronization method presented in Algorithm 1 has a provable bound, but it requires all the nodes to participate in one single synchronization session. This can be mitigated using a hierarchical approach. More specifically, if the network can be organized into clusters, we propose to synchronize the whole network using Algorithm 2 (Li & Rus, 2006). In Algorithm 2, the same method as in Algorithm 1 is firstly used to synchronize all the cluster heads by designing a message path that contains all the cluster heads (they are called the initiators base). Then, in the second step, the nodes in each cluster can be synchronized with their head.

Algorithm 2 The Cluster-Based Synchronization Algorithm

```
1: Run any clustering algorithm to organize the network into clusters
2: Synchronize the cluster heads with a base using Alg. 1
3: for each cluster do
4:  Synchronize the cluster members with the cluster head
```

This method can adapt to different clustering schemes. A cluster can be composed of the nodes within the transmission range of the cluster head; it can also be comprised of the nodes within some geographical area called a zone. For the first type of clustering, synchronization can be done with RBS. First, a reference broadcast is sent by the head to synchronize all the other cluster members. Then, any other node in the cluster sends out another reference broadcast to synchronize. The clock difference can be calculated with these two broadcasts and all the non head members can adjust their clocks according to the head's clock. In a zone clustering, the same method as Algorithm 1 is used to first design a cycle that includes all the nodes of the cluster and synchronize them all. The head of the cluster will be the initiator of the intra cluster synchronization (Li & Rus, 2006).

5.3 Diffusion-based Synchronization

The previous presented methods (cluster-based or all-node-based synchronization) use global time information sent to all the nodes and are not scalable for very large networks. The initiating node may encounter failure and, thus, the approach is not fault-tolerant. The nodes that participate in the synchronization must execute the related code approximately at the same time, which may be too hard in a large system. Now a diffusion method that is fully distributed and localized is introduced. In this method, synchronization is done locally, without a global synchronization initiator. It can also be done at arbitrary points in time as opposed to the strict timing requirements of the previous methods (Li & Rus, 2006).

The diffusion method achieves global synchronization by spreading the local synchronization information to the entire system. The algorithm can choose various global values to synchronize the network provided that each node in the network agrees to change its clock reading to the consensus value. An easy option is to choose the highest or lowest reading over the network. Synchronization to the highest or lowest value entails a simple algorithm (Li & Rus, 2006).

However, if there are faulty or malicious nodes, such a node may impose an abnormally high or low clock reading, which is likely to ruin the synchronization. To make the

algorithms more robust and reasonable, the following algorithms use the global average value as the ultimate synchronization clock reading. The main idea of the algorithms is to average all the clock time readings and set each clock to the average time. A node with high clock time reading diffuses that time value to its neighbors and levels down its clock time. A node with low time reading absorbs some of the values from its neighbors and increases its value. After a certain number of rounds of diffusion, the clock in each sensor will have the same value (Li & Rus, 2006).

There are two typical basic operations in diffusion-based synchronization scheme: 1) the neighboring nodes compare their clock readings at a certain time point and 2) the nodes change their clock accordingly. This, however, may be a problem because the clock comparison and the clock update cannot be done simultaneously (especially when clock comparison may take several steps). The clock updates based on the clock readings of the comparison time will be incorrect. The solution is to ask each node to keep a record of how much time elapses after the clock comparison on each node and use this time in the clock update (Li & Rus, 2006).

5.3.1 Synchronous Diffusion

Algorithm 3 (Li & Rus, 2006) shows the diffusion method. Synchronization between a sensor node and its neighbors is done by clock comparison and update operations. Because this algorithm only consider the time difference between two sensor nodes instead of the absolute clock time value, it is not required that all the sensors must do this local synchronization at the same time. In line 6, the exchanged value between sensor n_i and its neighbor n_j is proportional to the time difference between them.

Algorithm 3 Diffusion Algorithm to synchronize the whole network

1: **for** each sensor n_i in the network **do**
2: Exchange clock times with $n_i's$ neighbors
3: **for** each neighbor n_j **do**
4: Let the times of n_i and n_j be c_i and c_j
5: Change $n_j's$ time to $c_i + r_{i,j}(c_i - c_j)$
6: Change $n_i's$ time to $c_i - \sum r_{i,j}(c_i - c_j)$

5.3.2 Asynchronous Diffusion

In the previous section, a synchronous diffusion-based algorithm is presented. The synchronous algorithm is localized, but it requires a set order for all the node operations. In order to remove this constraint, the extension of the diffusion synchronization algorithm is here introduced. In this algorithm, all the nodes can perform operations in any order as long as each node is involved in the operations with nonzero probability. The following asynchronous averaging algorithm (Algorithm 4) (Li & Rus, 2006) gives a very simple average operation of a node over its neighbors. Each node tries to compute the local average

value directly by asking all its neighbors about their values; it then sends out the computed average value to all its neighbors so they can update their values.

Algorithm 4 Asynchronous Averaging Algorithm in a Sensor Network

1: **for** each sensor n_i with uniform probability **do**
2: Ask its neighbors the clock readings (read values from n_i and its neighbors)
3: Average the readings (compute)
4: Send back to the neighbors the new value (write values to n_i and its neighbors)

6. FAD(Fast Asynchronous Diffusion) Scheme

Several time synchronization algorithms have some problems when the algorithms escape their assumption and disconnection occurs in their network topology. For example, hierarchical topology has severe disadvantage when the network connection is broken. That is, all sensor nodes have to reorganize network connection and then time synchronization should be performed. Hence, asynchronous diffusion algorithm suggests new operation for global time synchronization among all the nodes in sensor networks.

$$C_{i-adjust} = \left(\frac{C_i + C_{j(neighbor)}}{2} \right) \tag{12}$$

In equation (12), $C_{i-adjust}$ presents an adjusted clock value, and $C_{j(neighbor)}$ is a clock value among neighbor sensor nodes. In asynchronous diffusion algorithm, a node n_i might have several clock values adjusted by algorithm 4 since all sensor nodes are assumed to be connected. In this case, a node n_i adjusts its local clock with the most recently received average clock value among a series of average clock values.

6.1 FAD Algorithm

Recently J. Bae and B. Moon (Bae & Moon, 2009) proposed a Fast Asynchronous Diffusion (FAD) clock synchronization algorithm in order to improve the diffusion-based asynchronous averaging algorithm (Algorithm 4). In this section, the different points about comparing asynchronous diffusion algorithm with the proposed FAD algorithm are presented. In asynchronous diffusion algorithm (Algorithm 4), each node uses the most recently received average clock value for adjusting its local clock when getting a series of average clock values. Meanwhile, the proposed scheme takes the mean of a series of average clock values received from all the neighbors under threshold for fast convergence. That is, a node adjusts its clock value with the mean of its neighbors' average clock values. Consequently the proposed algorithm (Algorithm 5) converges faster than asynchronous diffusion algorithm. The idea of FAD algorithm is expressed in equation (13).

$$C_{i-adjust} = \left(\frac{\sum_{j=1}^{N} \frac{C_i + C_{j(neighbor)}}{2}}{N} \right) \tag{13}$$

[N = number of neighbors]

FAD algorithm assumes that all the nodes in network have the same topology as asynchronous diffusion algorithm, but FAD algorithm differs from asynchronous diffusion in the process of getting average values. In other words, asynchronous diffusion scheme assumes that operating event must occurs in regular sequence, which uses average value received most recently. However, FAD algorithm doesn't consider operating sequence since it uses all the received average values (Bae & Moon, 2009).

Algorithm 5 Fast Asynchronous Diffusion(FAD) Algorithm in Sensor Network

1: **for** each node n_i with uniform probability **do**
2: Ask its neighbors the clock reading (read values from n_i and its neighbors)
3: **if** neighbor's clock < threshold
Average the reading(compute)
Send back to the neighbors the new value (write values to n_i and its neighbors)
4: **else** drop the received value
5: Each node n_i performs average operation again with all adjusted values received from its neighbors (write value to n_i)

In comparing FAD scheme with asynchronous diffusion scheme, there is actually no big difference in that more operations are required when the number of data increases in the viewpoint of algorithm complexity. But threre is an essential difference in the number of rounds needed for convergence. In next section, this difference is presented with the results of NS-2 simulation. Actually FAD has less number of rounds than asynchronous diffusion until convergence achievement is done. That is, FAD converges faster than asynchronous diffusion scheme. Generally, FAD seems to show less performce in the aspect of energy efficiency because FAD spends more time than asynchronous diffusion in getting average value. However, the time for synchronizing all the nodes in a sensor network is reduced since FAD achieves faster time synchronization than asynchronous diffusion scheme.

6.2 Performance Evaluation

The FAD scheme (Algorithm 5) is evaluated with NS-2 simulator (version 2.30) based on IEEE 802.15.4 module. The parameters such as propagation delay, collision, packet loss, and so on are considered. The simulation also includes asynchronous diffusion scheme for comparing FAD scheme with it. The simulation for time synchronization algorithms is performed within relative error of 0.01, and all nodes are assumed to have uniform distribution. The detail simulation parameters are summarized in Table 2.

Parameter	values
Number of Nodes	75, 90, 100, 125, 150, 200, 300, 400, 500
Sensor Field	100m × 100m
Transmission Range	15m
Physical Layer & MAC Layer	802.15.4
Routing Protocol	AODV
Relative Error	0.01
Uniform Probability (Mean)	0.5
Threshold (Percentage of Drift)	100%, 80%, 60%, 40%

Table 2. The Parameters for Simulation

6.3 Results and Discussions

In this simulation, the round is the number of the given algorithm performed at once. The number of operation is the sum of average operation from all nodes. In more detail, the operation ranges between zero and number of nodes participating in one round, and threshold is drift rate between received clock value and local clock value in one tick. Fig. 9 represents the comparison between asynchronous diffusion (left) and FAD (right) in the number of rounds with threshold value 40%. In this figure, the number of rounds decreases when the number of nodes increases. Each data point (*) represents the number of rounds when relative error becomes 0.01, and a line represents average value in each simulation condition.

Under this simulation, when the number of nodes is 500, FAD achieves time synchronization in average 31.7 rounds while asynchronous diffusion achieves it in average 35.8 rounds. When the number of sensor node is under 175, the time efficiency of FAD is better by 19% than asynchronous diffusion. When the number of sensor nodes is over 175, the time efficiency of FAD is better by 12% than asynchronous diffusion.

Fig. 10 shows the comparison between asynchronous diffusion (left) and FAD (right) with threshold value 40% in the number of operations. This figure represents that there is no big difference between FAD and asynchronous diffusion, and the number of total operations increases when the number of nodes increases. The reason can be explained from the results in Fig. 9. The number of rounds has exponential shape even thought the number of wireless sensor nodes increases. It means that these algorithms have to operate even though some additional rounds are not related with increasing the number of sensor nodes. That is, when the number of nodes is especially over the specific value, the number of rounds for time synchronization are not related with the number of nodes. Moreover, the number of operations increases when the number of nodes increases since the number of rounds is similar.

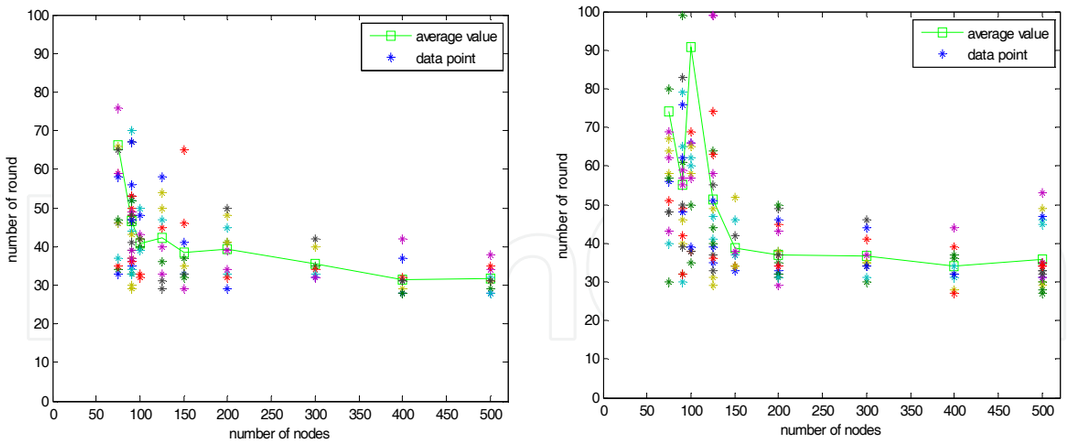


Fig. 9. Comparison between asynchronous diffusion (left) and FAD (right) in the number of rounds with threshold value 40%

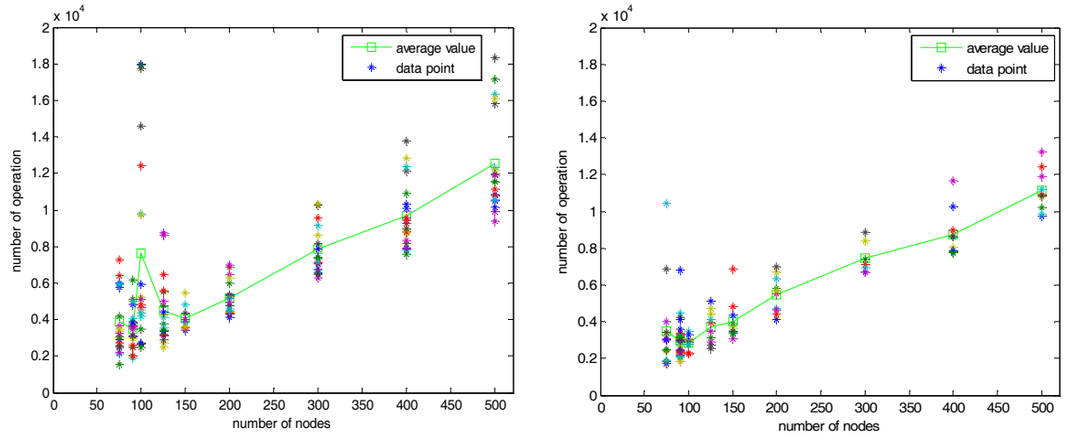


Fig. 10. Comparison between asynchronous diffusion (left) and FAD (right) with threshold value 40% in the number of operations

Fig. 11 represents the comparison between asynchronous diffusion and FAD in the average number of operations (left) and the average number of rounds (right) with threshold value(log scale). Fig. 11 (left) depicts the number of average operation in this simulation. Fig. 11 (right) shows average value of rounds. When the number of nodes is over 175, FAD uses the fewer number of operations than asynchronous diffusion. When the number of nodes is over 175, it is impossible for this simulation to compare FAD with asynchronous diffusion. However, when the number of nodes is under 175, FAD has better performance than asynchronous diffusion.

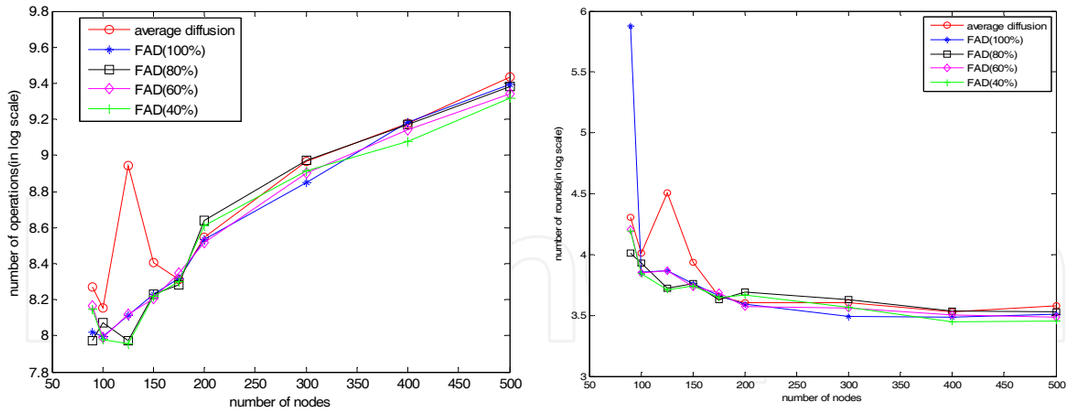


Fig. 11. Comparison between asynchronous diffusion and FAD in the average number of operations (left) and the average number of rounds (right) with threshold value(log scale)

7. Conclusion

Time synchronization is very useful function for improving device performance in WSN. In this chapter, we investigated time synchronization algorithms in WSN. Even though many algorithms are proposed until now, the best solution doesn't seem to exist since diversity devices are used in WSN. For the future research, meanwhile, time synchronization among heterogeneous devices will be new challenges.

8. References

Bae, J., & Moon, B. (2009). Time Synchronization with Fast Asynchronous Diffussion in Wireless Sensor Network, *International Conference on Cyber-enabled Distributed Computing and Knowledge Discovery (CyberC 2009)*, China, Zhangjiajie, October

Dolev, D.; Halpern, J., & Strong, H. R. (1984). On the Possibility and Impossibility of Achieving Clock Synchronization, *Proc. ACM Symp. Theory of Computing (STOC)*, May

Elson, J. & Estrin, D. (2001). Time Synchronization for Wireless Sensor Networks, *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS),Workshop on Parallel and Distributed Computing Issues in Wireless and Mobile Computing*, San Francisco, California, USA, April

Elson, J.; Girod, L. & Estrin, D. (2002). Fine-Grained Network Time Synchronization Using Reference Broadcasts, *Proc. Fifth Symp. Operating System Design and Implementation(OSDI 2002),* Dec.

Ganeriwal, S.; Kumar, R. & Srivastava, M. (2003). Time Sync Protocol for Sensor Network, *The First ACM Conference on Embedded Networked Sensor System (SenSys),* Los Angeles, Nov., pp. 138–149.

Ganeriwal, S.; Pöpper, C., Čapkun, S. & Srivastava, M. (2008), Secure Time Synchronization in Sensor Networks, *ACM Transactions on Information and System Security (TISSEC)*, Vol.11, no.4, July, ISSN:1094-9224

Girod, L. & Estrin, D. (2001). Robust range estimation using acoustic and multimodal sensing, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2001)*, March

Halpern, J.; Simons, B. & Strong, R. (1984). Fault-Tolerant Clock Synchronization, *Proc. ACM Symp. Principles of Distributed Computing (PODC)*, Aug.

Hofmann-Wellenhof, B.; Lichtenegger, H., & Collins, J. (1997). Global Positioning System: Theory and Practice, *4th ed. Springer Verlag*.

Intanagonwiwat, C.; Govindan, R., Estrin, D., Heidemann, J. & Silva, F. (2003). Directed Diffusion for Wireless Sensor Networking, *IEEE Trans. Networking*, vol.11, no.1, pp.2–16, Feburuary

Kopetz, H., & Ochsenreiter, W. (1987). *Clock* Synchronization in Distributed Real-Time Systems, *IEEE Transactions on Computers*, C-36(8), p.933–939, August

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM*, vol.21, no.7, pp.558–565

Lamport L. & Melliar-Smith, P. M. (1985). Synchronizing Clocks in the Presence of Faults, *J. ACM*, vol.32, no.1, pp.52-78, Jan.

Levine, J. (1999). Time synchronization over the internet using an adaptive frequency-locked loop, *IEEE Trans. Ultrason., Ferroelectr., Freq. Contr.*, vol.46, no.4, pp.888–896, Jul.

Li, Q., & Rus, D. (2006). Global Clock Synchronization in Sensor Network, *IEEE Trans. Computer Society,* vol.55, pp.214-216, Feb.

Lundelius, J. & Lynch, N. (1984). A New Fault-Tolerant Algorithm for Clock Synchronization, *Proc. ACM Symp. Principles of Distributed Computing (PODC)*, pp. 75-88, Aug.

Mannermaa, J; Kalliomaki, K., Mansten, T. & Turunen, S. (1999). Timing performance of various GPS receivers, *Proceedings of the 1999 Joint Meeting of the European Frequency and Time Forum and the IEEE International Frequency Control Symposium*, pp.287–290, April

Maroti, M.; Kusy, B., Simon, G. & Ledeczi, A. (2004). The flooding time synchronization protocol, *Proceedings of the ACM Conference on Networked Sensor Systems (SenSys'04)*, ACM Press, New York, pp.39–49

Mills, D. L. (1991). Internet Time Synchronization: The Network Time Protocol, *IEEE Transactions on Communications*, COM 39, no.10, pp.1482-1493, October

Mills, D. L. (1998). Adaptive hybrid clock discipline algorithm for the network time protocol, *IEEE/ACM Transactions on Networking*, vol.6, no.5, pp.505–514, Oct.

Romer, K. (2003). Temporal Message Ordering in Wireless Sensor Networks, *IFIP MedHocNet*, Mahdia, Tunisia, June

Sichitiu, M. L., & Veerarittiphan, C. (2003). Simple, Accurate Time Synchronization for Wireless Sensor Networks, *IEEE Wireless Communications and Networking Conference (WCNC) 2003*, New Orleans, LA, USA, March, vol.2, pp.1266 – 1273

Simon, G.; Maroti, M., Ledeczi, A., Balogh, G., Kusy, B., Nadas, A., Pap, G., Sallai, J. & Frampton, K. (2004). Sensor network-based counter sniper system, *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (Sen Sys),* ACM Press, New York

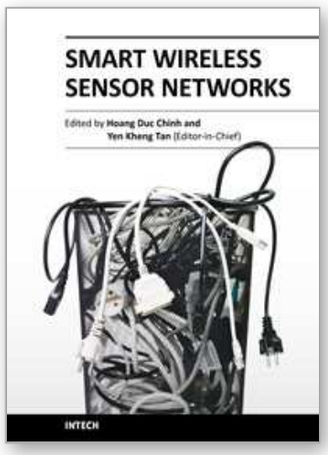
Sivrikaya, F. & Yener, B. (2004). Time Synchronization in Sensor Networks: A Survey, *IEEE Network*, vol.18, no.4, pp.45 – 50, July-Aug

Sommer, P. & Wattenhofer, R. (2009). Gradient clock synchronization in wireless sensor networks, *Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pp. 37-48,

Su, W. & Akyildiz, I. F. (2005). Time-Diffusion Synchronization Protocol for Wireless Sensor Networks, *IEEE/ACM Transactions on Networking,* vol.13, no.2, pp.384–397, April

Woo, A., & Culler, D. (2001). A Transmission Control Scheme for Media Access in Sensor Networks*, International Conference on Mobile Computing and Networking, (Mobicom)*, pp. 221–235, July

Yoon, S.; Veerarittiphan, C. & Sichitiu, M. L. (2007). Tiny-sync: Tight time synchronization for wireless sensor networks, *ACM Transactions on Sensor Networks (TOSN)*, vol., no.2, June



Smart Wireless Sensor Networks

Edited by Yen Kheng Tan

ISBN 978-953-307-261-6
Hard cover, 418 pages
Publisher InTech
Published online 14, December, 2010
Published in print edition December, 2010

The recent development of communication and sensor technology results in the growth of a new attractive and challenging area â€“ wireless sensor networks (WSNs). A wireless sensor network which consists of a large number of sensor nodes is deployed in environmental fields to serve various applications. Facilitated with the ability of wireless communication and intelligent computation, these nodes become smart sensors which do not only perceive ambient physical parameters but also be able to process information, cooperate with each other and self-organize into the network. These new features assist the sensor nodes as well as the network to operate more efficiently in terms of both data acquisition and energy consumption. Special purposes of the applications require design and operation of WSNs different from conventional networks such as the internet. The network design must take into account of the objectives of specific applications. The nature of deployed environment must be considered. The limited of sensor nodesâ€™ resources such as memory, computational ability, communication bandwidth and energy source are the challenges in network design. A smart wireless sensor network must be able to deal with these constraints as well as to guarantee the connectivity, coverage, reliability and security of networkâ€™s operation for a maximized lifetime. This book discusses various aspects of designing such smart wireless sensor networks. Main topics includes: design methodologies, network protocols and algorithms, quality of service management, coverage optimization, time synchronization and security techniques for sensor networks.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Jonggoo Bae and Bongkyo Moon (2010). Time Synchronization in Wireless Sensor Networks, Smart Wireless Sensor Networks, Yen Kheng Tan (Ed.), ISBN: 978-953-307-261-6, InTech, Available from: http://www.intechopen.com/books/smart-wireless-sensor-networks/time-synchronization-in-wireless-sensor-network

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820

Fax: +385 (51) 686 166
www.intechopen.com

Fax: +86-21-62489821

IntechOpen