Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2013, Article ID 532986, 10 pages
http://dx.doi.org/10.1155/2013/532986

Review Article

A Comparison of Clock Synchronization in Wireless Sensor Networks

Seongwook Youn

Computer Science Department, University of Southern California, 941 Bloom Walk, SAL 300, Los Angeles, CA 90089-0781, USA

Correspondence should be addressed to Seongwook Youn; seongwook.youn@gmail.com

Received 8 June 2013; Revised 20 November 2013; Accepted 21 November 2013

Academic Editor: Hongli Xu

The recent advances in microelectro devices have led the researchers to an area of developing a large distributed system that consist of small, wireless sensor nodes. These sensor nodes are usually equipped with sensors to perceive the environment. Synchronization is an important component of almost all distributed systems and has been studied by many researchers. There are many solutions for the classical networks, but the traditional synchronization techniques are not suitable for sensor networks because they do not consider the partitioning of the network and message delay. Additionally, limited power, computational capacity, and memory of the sensor nodes make the problem more challenging for wireless sensor networks. This paper examines the clock synchronization issues in wireless sensor networks. Energy efficiency, cost, scalability, lifetime, robustness, and precision are the main problems to be considered in design of a synchronization algorithm. There is no one single system that satisfies all these together. A comparison of different clock synchronization algorithms in wireless sensor networks with a main focus on energy efficiency, scalability, and precision properties of them will be provided here.

1. Introduction

Wireless sensor networks are the networks that consist of mobile wireless computing devices, in which these devices are usually equipped with sensors to perceive the environment. Along with the recent advances in technology and the increasing demand, sensor networks are now being widely used in many applications. Wireless sensor networks have many applications including environmental monitoring, health monitoring, inventory location monitoring, and objects tracking. Features of a sensor network, such as size (number of nodes), density, and connectivity, vary depending on the application. Sensor nodes in the network are mostly mobile devices equipped with limited power and computation capabilities. Hence, a reasonable ordering of events in such environments is a challenging task.

This paper examines the clock synchronization issues in ad hoc and sensor networks [1]. Clocks can be out of synchronization in two ways: shifting (clock offset or phase offset) or drifting (clock skew-oscillator's frequency offset). In the case of shifting, they run at the same frequency, but their clock readings differ by a constant value—the offset between the clocks. In the case of drifting, they run at different frequencies. Synchronizing drifting clocks is much more costly and difficult than synchronizing two shifting clocks. Clocks of nodes may run at slightly different frequencies, which is the main reason why clock offsets keep drifting away due to the imperfections in the quartz crystal. Adjusting clock skew can guarantee long-term reliability of synchronization and reduce the number of message exchanges. Some of the previous algorithms adjust the frequency offset (clock skew) relative to a certain frequency [2, 3]. Maggs et al. [4] proposed a consensus clock synchronization that provides internal synchronization to a virtual consensus clock. It is sensitive to the limited resources available to sensor nodes and is robust to many of the challenges faced in dynamic ad hoc networks.

Feedback-based synchronization (FBS) scheme to compensate the clock drift caused by both internal perturbation and external disturbance was proposed by Chen et al. [5]. It showed that FBS is much more robust than the delay measurement time-synchronization (DMTS) protocol. Misra and Vaish [6] suggested a reputation-based role assigning scheme for RBAC. The main objective of this scheme is to manage reputation locally with minimum communication

and delay overhead and to assign appropriate role or level to the deserved nodes in order to increase the throughput of overall network. Their scheme showed the increase in throughput by around 32% at the consumption of little more energy. Liu et al. [7] proposed a Kalman filter based advanced SCTS (ASCTS) mechanism. They proved the close relationship between the basic phase locked loop (PLL) employed by SCTS and the Kalman filter employed by ASCTS.

Many synchronization techniques have been proposed in literature for either central [8, 9] or distributed [10–12] systems. However, these classical synchronization techniques are not suitable for wireless sensor networks, since they do not take into account the partitioning of the network and the message delay.

There are mainly six requirements to be considered in design of a synchronization algorithm: energy efficiency, cost, scalability, lifetime, robustness, and precision. There is no one single system that satisfies all these together. The author will be providing a comparison of different clock synchronization algorithms in wireless networks along three axes: energy efficiency, scalability, and precision properties.

The paper is organized as follows. I will be first talking about the synchronization problem in the next subsection. Then, I will give an overview of traditional synchronization methods in Section 2. After a brief introduction to wireless sensor networks, challenges, design issues, sources of error, and requirements of a synchronization method for wireless sensor networks will be analyzed in Section 3. I will present four different synchronization methods and then compare them based on three aspects.

2. Traditional Synchronization Methods

Possible set of solutions to the problem depends on the system in use. For instance, the problem can be addressed by a centralized server in centralized systems. Two such systems are described by Cristian [8] and Gusella and Zatti [9]. The method presented in [8] depends on a central time server that is connected to an accurate time source like UTC (Coordinated Universal Time). To get the actual time, the client sends a request to the server. Receiving the request, the server prepares a response by appending its current clock time t to it and sends it to the client. Then the client adjusts its time as $t + rrt/2$, where rrt is the round trip time elapsed for the message to travel from and then back to a sender.

Rather than the clients asking for time as in [8], a time server polls the machines periodically in [9]. The procedure starts with the server requesting for current clock times of the clients. Once all the responses are received, the server finds the time of each client by using the round trip times of the messages, a method similar to [8]. The actual time is then calculated in the server by averaging those values as well as its own time. Instead of sending the calculated time, the server sends to each client the amount of time that it should adjust. The idea behind this is to avoid the errors introduced by the round-trip time estimations.

The clock synchronization problem gets more complicated in distributed systems due to lack of a global clock. There are two concepts to be considered in this case, either to synchronize the physical clock or the logical clock. In physical clock synchronization, the goal is to bring together the physical clocks of each machine to a very similar point; whereas in logical clock synchronization, gravity is the accurate ordering of relevant events.

Network Time Protocol (NTP) is the most commonly used method on the Internet for physical clock synchronization [12]. NTP is a layered client-server architecture based on UDP message passing. It operates with a hierarchy of levels, where levels are assigned a number called the stratum. At the lowest level are the stratum 1 (primary) servers, which are directly synchronized to national time services. In the next level, there are the stratum 2 (secondary) servers that are synchronized to stratum 1 servers. And the hierarchy continues the same way until the highest level.

In some distributed systems, it is more important to have a consistent and logical ordering of events, rather than knowing the actual occurrence time for each individual event. For such systems, it is not required to have absolute clock synchronization as it was the case in physical clock synchronization. Lamport [11] and Fidge [10] are the two most remarkable solutions for logical clock synchronization in distributed systems.

Lamport [11] defines an ordering of events using the concept of causality. If an event a could have affected the outcome of event b, then it is referred as event a "happened before" event b. The partial ordering of events is discussed in the paper, which is obtained by the "happened before" relation. For the partial ordering, there are two rules to be considered. The first is to increment the local clock between any two successive local events. The second is, upon receiving a message from another process with a local timestamp t of that sending process, to set the local clock greater than or equal to the maximum value of either t or the local clock value. Finally, they use these logical clocks to obtain total ordering across all processes and events.

Fidge [10] also defines a partial ordering of events using the causality concept. However, rather than using a single value for each timestamp, they choose to use a vector of values. The vector is initially set to $(0, 0, \ldots, 0)$, where each index corresponds to a processor. In case of a local event at processor P_i, the value at index i is incremented. When a processor P_i receives a message from processor P_j with timestamp vector T, P_i sets the time in each index to maximum value of either the corresponding value of T or the local vector value. The advantage of keeping a vector of timestamps and maximizing it among processors is that it allows ordering not only the events within a process, but also the events in different processes.

3. Synchronization Issues in Wireless Sensor Networks

Sensor networks are the networks that consist of mobile wireless computing devices that are equipped with sensors to perceive the environmental conditions, such as temperature, pressure, and humidity. The traditional synchronization techniques described above are not suitable for such networks.

In this section, I will first talk about the sensor networks in general and then discuss the main challenges of synchronization in sensor networks, which involves desired properties and design principles of a synchronization scheme, as well as main possible resources of error.

Although the initial settings might be the same, real clocks at different computing devices can be different due to some variances in the counting rates of the clocks. For some node i in the network, a hardware oscillator assisted computer clock installed in that node implements an approximation $C(t)$ of real-time t as

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

where a_i denotes the angular frequency (or rate) of the hardware oscillator and b_i denotes the difference to the real time t. In literature, this angular frequency is usually referred as clock drift or skew and the difference as clock offset [13]. In a perfect case, the rate of a clock (dC/dt) would be equal to zero. However, due to the environmental conditions, such as temperature, pressure, and humidity, the clocks are subject to some drift with a maximum value of ρ such that

$$1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho. \tag{2}$$

Although having different values for different computers, today's clock hardware typically provide a value of 10^{-6} for ρ [14], which means the clocks drift away from each other by at most one second in ten days. Behaviors of clocks with different dC/dt values are the standard timescale used by most of the nations in the world, which is based on the Earth's rotation about its axis [15].

Clock synchronization problem deals with ways of bringing the clocks of different computers (or processes, devices...) close to each other by communicating among them. More precisely, the clock synchronization problem aims to equalize $C_i(t)$ for all the nodes $i = 1, \ldots, n$ or some subset of nodes in the network. Adjusting the clock values for once is not enough, since the clocks will be drifting away again later. Hence, one can choose to either equalize the rates along with the offset or apply the synchronization repeatedly.

3.1. Sensor Networks. The recent advances in small electro devices have aroused interest in the development of large and distributed systems of small, wireless sensor nodes that communicate with each other. These sensor nodes mostly consist of components with sensing, data processing, and communicating capabilities. Although being limited in power, computational capacities, and memory, they can be used collaboratively to monitor the environmental conditions. Hence, a sensor network is network that is composed of a large number of spatially distributed sensor nodes that monitor physical or environmental conditions.

Types of sensors in a sensor network include low sampling rate magnetic, thermal, visual, infrared, acoustic, and radar. These sensors are able to monitor a wide variety of ambient conditions such as temperature, humidity, vehicular movement, lightning condition, pressure, soil makeup, noise levels, the presence or absence of certain kinds of objects, mechanical stress levels on attached objects, and speed, direction, and size of an object [16]. Initially, the development of wireless sensor networks originated by military-oriented applications (i.e., monitoring forces, battlefield surveillance). However, today wireless sensor networks are being used in various domains for many other applications. Some of these domains and sample applications in those domains have been listed below.

(i) Military: battlefield surveillance, targeting, monitoring forces, equipment and ammunition, and battle damage assessment.

(ii) Environmental: fire, flood, earthquake detection, and biocomplexity mapping.

(iii) Health: tracking and monitoring doctors/patients in a hospital, human physiological data telemonitoring.

(iv) Scientific: space and undersea exploration, cosmic radiation, and nuclear reactor control.

(v) Home: home automation, smart environment design.

(vi) Commercial: virtual keyboards, monitoring product quality, interactive museums, and detecting and monitoring car thefts.

Depending on the application, the number of nodes in a sensor network can be in the order of hundreds or thousands. These nodes are usually inaccessible and unattended, making the network topology prone to dynamic changes. Thus, robustness and self-configuration are the important requirements to be considered in design of a sensor network. Energy efficiency is another important concern for wireless sensor networks, since nodes are often inaccessible and have small sizes, which causes them to possess or produce limited power.

3.2. Desired Properties. In this section, the main requirements of a synchronization method for wireless sensor networks are listed and discussed. There is a trade-off among each of these features, and no one single system satisfies all these together [3, 13, 17].

(i) Energy efficiency: sensors in a wireless network are small and untethered devices. Hence, a synchronization scheme should take into account the limited energy resources and utilize energy in an efficient way.

(ii) Scalability: sensor networks usually consist of hundreds to thousands of nodes. Hence, a synchronization scheme should be able to scale well with increasing node density or number of nodes.

(iii) Precision: refers to how much the local clocks differ from either each other or an external standard clock. Desired precision can range from milliseconds to seconds depending on the application. For some applications, it is enough to only have a reasonable ordering of events, whereas for others a very high precision might be required.

(iv) Robustness: sensors in the network are usually mobile and untethered. There is great chance for a node

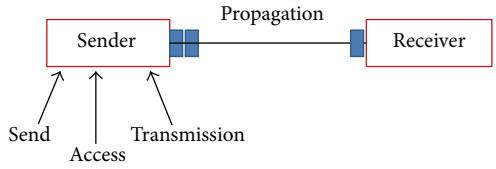

Figure 1: Sources of delay while transmitting a packet from a source to a destination in the wireless sensor network.

to fail or left unattended for a long time. Hence, robustness to such failures is a desired property for a synchronization method.

(v) Lifetime: amount of time for synchronization to last. Depending on the scheme, it might be either instantaneous or as long as the network operates.

(vi) Scope: for some applications, it is enough to synchronize only a subset of the network at a time, whereas for others a global synchronization might be required. Scope defines the geographic span of nodes that need to be synchronized.

(vii) Cost: sensor nodes are usually small and low cost devices. So, it is not a reasonable thing to equip a node with expensive hardware, such as a GPS receiver. Cost can play an important role for the overall system, considering that the number of sensors can get extremely large.

3.3. Main Sources of Errors. In this section, main possible sources of error in a synchronization algorithm are presented. When two nodes want to synchronize, they need to communicate with each other by message exchange. However, there are different types of delays on the path from the sender of a message to the receiver that cause errors in clock estimations. Figure 1 explains a schematic representation of where these delays happen. In the remaining, I discuss each error source individually.

(i) Send time: corresponds to the time spent in the sender node for constructing the packet at the application layer and sending it to the MAC layer. This time depends on the operation system being used, hence causing a nondeterministic delay on sender.

(ii) Access time: the time spent at the MAC layer waiting for access to the transmission channel. This delay plays an important role for most of the systems.

(iii) Transmission time: corresponds to the time taken for a message to be transmitted on the wireless link. This is a deterministic delay and can be estimated by the length of the message and the speed of the radio.

(iv) Propagation time: this is the time spend on the wireless link from sender to the receiver, once the packet leaves the sender. This delay is also deterministic and depends on the distance between the nodes.

(v) Reception time: this refers to the time spent on the receiver for receiving the packet and passing it to the MAC layer. This time corresponds to the transmission time on the receiver side and can be estimated in a similar manner.

(vi) Receive time: corresponds to the time for processing the incoming packet at the receiver and sending it to the application layer. This time can be thought as dual of send time at the sender.

3.4. Design Principles. In [3], Elson and Romer discussed five main design properties of a wireless network synchronization algorithm. The first principle is a multimodal, tiered, and tunable design. As mentioned in Section 3.2, there is always a trade-off among the desired attributes of an algorithm. According to this, the first principle says that synchronization should contain different models with different attributes, so that one can tune it by changing a set of parameters for different applications.

The second principle offers each node in the network to store relative drift and phase information locally, rather than keeping a global timescale. This kind of design purveys the error to be dependent on the distance between the nodes, not the distance to a master clock.

The third principle is postfacto synchronization, which has been widely used by many algorithms [2, 17, 18]. Postfacto synchronization offers the local node clocks to run asynchronous until the timestamps of different clocks need to be compared. This provides a lot of energy savings by forcing the resources to be used only when required. Final two principles involve being adaptable to different applications and exploiting the domain knowledge.

Apart from these, there are certain other concerns that should be taken into account for the design. I will be mentioning only two of them that I will give better understanding for further reading of this paper and refer the reader to [3, 15] for more detailed analysis. First issue is single-hop versus multihop synchronization. Most of the traditional methods assume that all the nodes in the network can communicate with each other or the network topology has lowlatency. However, for sensor networks, this may not be always the case that there might be more than one broadcast domains. Nodes in different domains can communicate with each other via routers that appear in both domains (at the intersection of two domains).

Second important concern is static versus dynamic network topology. Sensors are usually moving devices in the network. Hence, the network topology is subject to change frequently. Also, due to limited power or range of sensor, it is possible to have link failures any time in the network. A synchronization scheme should be able to adapt these changes dynamically.

4. Methods for Synchronization in Wireless Sensor Networks

Four different synchronization methods are presented in here. These methods are selected because either they are one of the first systems proposed for sensor networks or most widely used/referred systems.

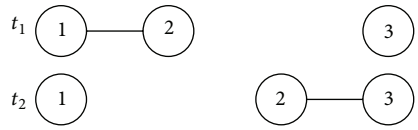

Figure 2: Network topology and message flow in ad hoc networks.

4.1. Synchronization in Ad Hoc Networks. Römer [14] proposed a time synchronization algorithm for ad hoc networks. In ad hoc networks, nodes are usually mobile and have limited communication range. Thus, the network topology is prone to frequent changes. This has been depicted in Figure 2 by an example. At time t_1, only nodes 1 and 2 are within communication range of each other. Then, node 2 moves closer to node 3; hence, at time t_2, only nodes 2 and 3 can communicate. There is no time between t_1 and t_2, in which nodes 1 and 3 can communicate to each other directly or indirectly. In contravention of this, nodes 1 and 3 can communicate with each other in a unidirectional way: at time time t_1 node 1 sends a message to node 2, which is stored at node to and then forwarded to node 3 at time t_2.

Traditional methods assume that nodes in the network can send messages to each other periodically and the round trip time between two nodes can be estimated. However, these assumptions no longer hold for ad hoc networks. According to Römer, an ad hoc network synchronization algorithm should not require a particular network topology and be able to handle all kinds of partitioning. This in mind, Römer, makes two assumptions about the network. First assumption is that the maximum clock drift ρ_i is known for all computer clocks. Secondly, it is assumed that if two adjacent nodes start to communicate with each other, then the connection lasts long enough to allow the two nodes to exchange one more (additional) message.

The main idea of the proposed algorithm is to transform the timestamp generated by an untethered local clock of a sending node to the local clock of the receiver node. According to this, if a node wants to send a message to another node in the network, it creates a timestamp using its own local clock and attaches it to the message. When the message is received by the other node, the timestamp is first transformed from local time of sender to UTC and then from UTC to the local time of the receiver. Due to various reasons, such as unpredictability of computer clocks, these transformations cannot be done exactly. So, the algorithm uses lower and upper bounds for the interval of the exact time. The relationship between the computer clock difference ΔC and the real time difference Δt can be given as

$$1 - \rho \leq \frac{\Delta C}{\Delta t} \leq 1 + \rho, \tag{3}$$

which can be transformed into

$$\begin{aligned} (1-\rho)\,\Delta t \leq \Delta C \leq (1+\rho)\,\Delta t \\ \frac{\Delta C}{1+\rho} \leq \Delta t \leq \frac{\Delta C}{1-\rho}. \end{aligned} \tag{4}$$

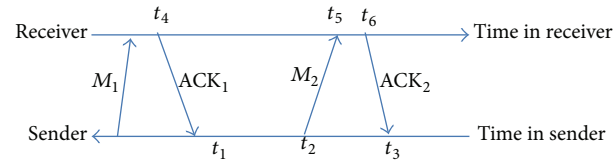

Figure 3: Estimation of message delay.

These lead to the consequence that the computer clock difference ΔC lies within $[(1-\rho)\Delta t \leq \Delta C \leq (1+\rho)\Delta t]$. Similarly, the interval for real time difference Δt can be given by $[\Delta C/(1+\rho), \Delta C/(1-\rho)]$. Let ρ_s and ρ_r be the maximum clock drifts of the sender and the receiver nodes, respectively. Then, the algorithm runs as follows.

(i) Sender node s generates a message with its local timestamp.

(ii) When the receiver node r gets the message, it first estimates the computer clock difference ΔC from the real time interval as $[\Delta C/(1+\rho_s), \Delta C/(1-\rho_r)]$.

(iii) Then, the receiver calculates the computer clock difference relative to its local time as $[\Delta C((1-\rho_r)/(1+\rho_s)), \Delta C((1+\rho_s)/(1-\rho_r))]$.

There is the message delay d that the transformation algorithm needs to take into account in order to find these intervals exactly. However, this delay is not constant for all message exchanges. Therefore, they choose to estimate a delay interval for each message independently. The delay for message M_2 in terms of the receiver's clock using two consecutive message exchanges can be given by

$$0 \leq d \leq (t_5 - t_4) - (t_2 - t_1)\,\frac{1-\rho_r}{1+\rho_s}. \tag{5}$$

However, this estimation has two disadvantages. First, the time between two consecutive messages can be quite high, resulting in the values for $(t_5 - t_4)$ and $(t_2 - t_1)$ to be large. Second, the delay is calculated by using two different message exchanges. This requires keeping track of state information in case of multiple message transfers among many other nodes. Römer proposes to avoid the first disadvantage by sending dummy messages when if these values sum up to be large. For the second case, he offers to delete the state information at the cost of a later dummy message. The estimation process of message delay is shown in Figure 3.

The algorithm described above works for the message exchanges between two adjacent nodes in the network. However, as mentioned before, the message exchanges between two nodes can be unidirectional and delayed. For a message to be transfered from node 1 to node n with message exchanges in the sequence of nodes $1, 2, \ldots, n$, the round trip time between each pair of nodes and the idle time of the node are also maintained to be used in the time calculations.

Once the time intervals, $[t_1, t_2]$ and $[t_3, t_4]$, for two different events are computed as described above, finding whether either one of them happened before the other can be done by comparing these intervals; that is, $[t_1, t_2]$ ¡ $[t_3, t_4]$.

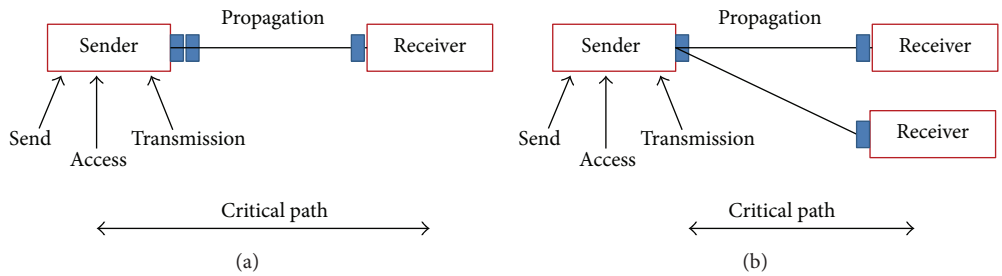

Figure 4: (a) Critical path for traditional methods. (b) Critical path for RBS.

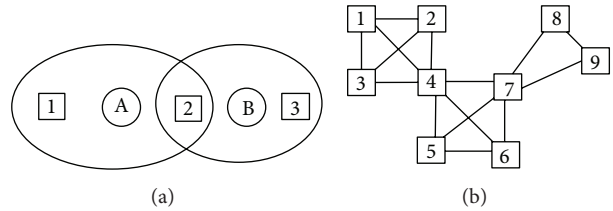

Figure 5: (a) A sample topology with two broadcast domains. (b) A logical topology.

The answer is either "yes" if $t_2 \text{ ¡ } t_3$ or "no" if $t_4 \text{ ¡ } t_1$ or "maybe" otherwise. Also, the distance between these two intervals can also be given as

$$\left|[t_1, t_2] - [t_3, t_4]\right| \leq \frac{\left(\max\left(t_4, t_2\right) - \min\left(t_3, t_1\right)\right)}{\left(1 - \rho\right)}. \quad (6)$$

The prototype experiments presented in the paper show that the performance of the system decreases in proportional to the increase in timestamp intervals, which may be caused by either the age of the timestamp or the number of hops used to pass the message.

4.2. Reference Broadcast Synchronization (RBS). Elson et al. [2] present a synchronization scheme called Reference Broadcast Synchronization (RBS) for wireless sensor networks. The main idea of the algorithm lies under the assumption that when a node broadcasts a reference beacon to its neighbors, the receivers will get the message approximately at the same time. As opposed to the traditional synchronization methods that try to synchronize the time between the sender and receiver nodes, they propose to synchronize a set of receivers with each another. The behind principle for doing this is to remove the errors caused by the sender's non-determinism. Elson defines the critical path in a message exchange to be the path from the sender node to the receiver node that includes all sources of error (send, access, transmission, propagation, reception, and receive delays) as explained earlier in Section 3.2. Since RBS takes into account only the arrival times of a message in each receiver, the three error sources (send, access, and transmission delays) are eliminated by default. Figure 4 shows the critical path of traditional methods and RBS.

The algorithm estimates the phase offset between two receiver nodes i and j as follows. When the sender broadcasts m reference messages, the n receivers record the arrival time of the message according to their local time. After the receivers exchange the reordered times, the receiver i can compute the phase offset between the receiver j by

$$B_{i,j} = \frac{1}{m}\sum_{k=1}^{m}\left(T_{jk} - T_{ik}\right), \quad (7)$$

where T_{jk} refers to the time of node j at receiving the message k. The right side of the equation represents the average of all phase offsets between i and j for all m messages. This averaging provides better precision statistically.

To find the clock drift, they propose to perform a least-squares linear regression over the phase offsets of all exchanged messages. This implies fitting a best line to all the phase offset observations of two receiver nodes over time. The slope and the intercept of the fitted line then provide the drift and the offset of one node with respect to the other. Such an approach allows the relative clock values to be computed even in case of missing messages.

As explained, the algorithm finds the relative clock offset and the drift from over multiple message exchanges. This provides postfacto synchronization by saving energy for the cases where synchronization is needed infrequently. When desired, nodes can turn on their power and transfer messages until the best fit line is computed reasonably. Storing the relative clock drifts and the offsets with respect to the other nodes, rather than correcting the local clocks according to a global time scale, also provides important energy savings.

Elson et al. [2] show that the proposed method can be generalized to clock synchronization in multihop networks. Consider the example on the left of Figure 5. The larger circles represent two different broadcast domains, rectangles represent nodes in corresponding domains, and A and B are the reference nodes. According to this topology, nodes 1 and 2 cannot communicate with each other directly. To compare the two events, e1 on node 1 and e3 on node 3, node 2 first

uses A's reference broadcast to convert the clock value of e1 to its own clock value. Similarly, using B's reference broadcast, node 2 converts this value to node 3's clock value. The final value can then be compared with e3, since both are based on node 3's clock.

The technique explained above can be extended to networks of more than two domains with multiple gateways. To do this, the network topology can be represented by a logical graph, in which there is a link between two nodes if they receive a common broadcast. See the graph on the right of Figure 5 for a sample logical topology graph. A series of conversions can be performed on this graph by finding a shortest path between the nodes. Also, the weights of the links can be used for representing the quality of the conversion.

4.3. Timing-Sync Protocol (TSPN). Ganeriwal et al. [18] propose a synchronization scheme called Timing-Sync Protocol for sensor networks (TSPN). One of the main concerns of the paper is to achieve high accuracy even for a large number of nodes being deployed. The algorithm presented is based on a sender-receiver synchronization approach. They argue that this classical approach gives better results than synchronizing only the receivers with each other (i.e., RBS [2]). The principle is that messages are time stamped at the MAC layer, which removes sources of error at the sender and the receiver.

There are two assumptions about the network that make the proposed algorithm work. First assumption is that every node in the network knows the set of nodes that it can communicate with. Secondly, they suppose that it is possible to create a spanning tree in the network by using the bidirectional links among the nodes. Based on these, there are two main steps of the algorithm, which involves "level discovery phase" and "synchronization phase."

The first step of the algorithm is the "level discovery phase" to create a hierarchical topology of the network. A root node is assigned at the level 0, and it initiates the phase by broadcasting a level discovery message. Nodes that receive this message assign themselves to one level greater than the received message. These nodes then broadcast a new level discovery message that contains the level of the broadcasting node. For instance, a node that receives a level discovery message from level 0 sets its level to 1 and then broadcasts another level discovery message affirming that it is from level 1. This scheme continues until every node in the network establishes a level.

The second step of the algorithm is the "synchronization phase," which involves a pairwise synchronization along the edges of the composed topology. This phase is also initiated by the root by broadcasting a time-sync message, and proceeds from the nodes of a lower level until the highest level. A node receiving a time sync from the root sends a synchronization-pulse message to the root indicating that it wants to adjust its clock to the root. The root responds back with an acknowledgment containing the required information for the node to synchronize. Nodes at level 2 will also be receiving the synchronization pulse message sent to the root, which will act as a time sync for these nodes. This hierarchical way of synchronizing from root up to the highest level of the topology induces every node in the network to be synchronized with the root node. Therefore, the root node is referred as a user node and is usually equipped with a GPS receiver.

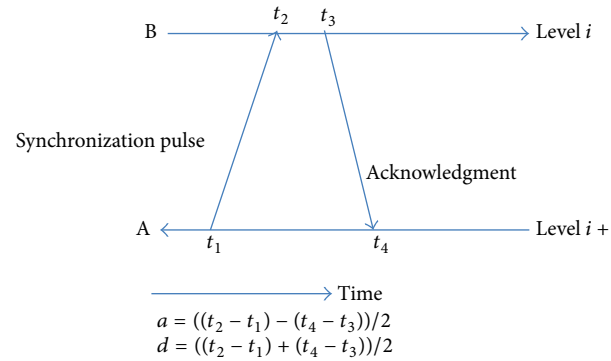

Figure 6: Message exchange between the nodes A and B.

Consider Figure 6, where node A from a higher level $i+1$ wants to synchronize its clock to a node B from a lower level i. At time t_1, A sends a synchronization-pulse message to node B, which contains the level number $i+1$ and time t_1. B receives this message at time t_2 and sends back an acknowledgment message at time t_3 including the values for t_1, t_2, t_3, and the level information i. When A gets the acknowledgment message at time t_4, it can calculate the clock drift a and propagation delay d as follows:

$$a = \frac{(t_2 - t_1) - (t_4 - t_3)}{2}$$
$$d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2}. \quad (8)$$

After finding a and d, A can now adjust its clock to the clock of B. Authors claim that postfacto synchronization can be applied for energy savings in multihop networks. Assume that node A needs to send a message to node E through the path A-B-C-D-E. Then, synchronization is started between two nodes, once the there is a message exchange. For instance, when B gets a message from A, it synchronizes with A first using TSPN and then sends the message to the next hop C.

4.4. Flooding Time Synchronization Protocol (FTSP). Maróti et al. [19] proposes the Flooding Time Synchronization Protocol (FTSP) to achieve network-wide time synchronization. The goal of the scheme is to provide high accuracy and scalability under large number of mobile nodes. For better accuracy, MAC layer time stamping is utilized to eliminate most of the error resources on the critical path from sender to receiver.

Two of the error resources mentioned in Section 3.2—transmission and reception delays—are analyzed in more detail and divided further into four categories: interrupt handling time (the delay between and microcontroller), encoding time (time to encode and transform message to electromagnetic waves), decoding time (time to decode and transform message from electromagnetic waves to binary data), and byte alignment time (delay from different alignment of sender and receiver). The two triangles at the top and bottom show

the time when the message is time-stamped. This time-stamping mechanism eliminates the jitter of interrupt handling time on the sender.

FTSP further removes other sources of error except the propagation delay as follows. A broadcast message is time stamped both at the sender and the receiver. A message consists of four regions: preamble bytes, sync bytes, actual data with a descriptor, and crc bytes. While preamble bytes are transmitted, the receiver radio can synchronize to the carrier frequency of the message signal. Bit offset is calculated from sync bytes at the receiver for bit alignment, and the message is time stamped after sync bites are sent/received. Finally, the clock drift is found from a best line fitted on data points representing time and clock offset by linear regression (a similar approach applied in TSPN).

For multihop synchronization, FTSP uses reference points that hold both the local and global timestamp of an instance of time. All the nodes in the network synchronize to a root node. If a node can communicate with the root directly, then it will collect reference points from the root to synchronize. However, if it is not in the communication range of the root, then it can get the reference points from other synchronized nodes in the network. The root node is selected dynamically according to the changing network topology.

5. Comparison of Methods

5.1. Precision. Precision designates the accuracy of the algorithm and can refer to either absolute (with respect to an external standard clock) or relative (with respect to nodes within a network). Römer [14] conducts their experiments on a prototype system and reports 3ms inaccuracy on a test set of 5 hops. Their accuracy degreases as the age of time stamp and the number of hops increase. This number is the lowest precision among all other three methods in consideration. This is mainly due to two reasons. First reason is that Römer's method takes into account only the current and one previous message exchange between two nodes. Although this reduces the overall system complexity, the accuracy degrades from all the error sources (send, access, transmission, propagation, reception, and receive delays) on the critical path from sender to receiver. Unlike RBS and FTSP, no averaging or best line fitting on the previous offset points is applied. This causes Römer's method also to be sensitive to failures. Second reason is that Römer does not exploit the MAC layer time stamping. As analyzed in [18, 19], time stamping messages at the MAC layer remove considerable amount of error sources on the critical path. Recall that TPSN does not use previously exchanged messages but utilizes the MAC layer implementation.

RBS [2] follows a different approach than all other three methods in the sense that it synchronizes receivers with each other, rather than receiver to sender. This removes the nondeterminism at the sender (send, access, and transmission delays). With a 4-hop network on Berkeley motes, average error is reported as 3.68 μs. This is considered to a high precision. TSPN also conducts its experiments on Berkeley motes and reports an average error of 20 μs. Although this is worse than RBS, the authors of TSPN argue that this is due to the fact that RBS experiments were conducted on superior operating system. Under same circumstances, TSPN is claimed to be achieving two times better precision. Authors argue that this is due to two reasons: MAC layer utilization and two way message exchange between two nodes in TSPN.

Table 1: Synchronization error (in μs).

	Romer	RBS	TSPN	FTSP
Average error	207	29.13	16.9	0.95
Worst case error	274	93	44	4.32
Best case error	0	0	0	0

Table 2: Synchronization error over multihop.

Synchronization ad hoc network (Romer)	3 ms error on 5 hops
RBS	3.68 μs error on 4 hops
TSPN	20 μs error on 4 hops
FTSP	3 μs error on 6 hops

Despite these, if RBS was implemented with MAC layer utilization ability, it would give better results than TSPN under the scenario that the nodes synchronize with each other frequently and at constant time intervals. The reason is that averaging and line fitting for clock offset and drift in RBS will eventually dominate the two-way message exchange in TSPN statistically.

FTSP [19] is implemented on UCB Micra platforms and achieves 3 μs error on 6 hops, which is 0.5 μs error per hop. This is the best precision among all other three methods. There are two contributors for this result. First, FTSP exploits MAC layer time stamping more than TSPN does. It removes the sources of errors on the sender and receiver by a smarter implementation that adjusts the receiver of a message to the carrier frequency. Second, FTSP exploits from linear regression to estimate clock drift and offset, like RBS. But different than RBS, FTSP is a sender-receiver synchronization method that uses multiple time stamps (both global and local) corresponding to a reference point. Synchronization error statistics of each method shown in Tables 1 and 2 show the synchronization error in case of multihop of each method.

5.2. Energy Efficiency. Considering that sensors are small and untethered devices, energy resources should be utilized in an efficient manner. There are three issues to be considered: post-facto synchronization, computation, and communication. Post-facto synchronization proposes the nodes to stay in a low-power state with unsynchronized clocks, until a event of interest occurs. All three algorithms (except FTSP) suggest a post-facto synchronization scheme.

There is no quantitative data; however, Römer's methods behave the best in terms of energy utilization. Römer takes into account that a node can go idle for a long time; hence, the exchanged messages contain this idle time information to be used in clock drift calculation. Also, the computational complexity and message overload are very low. Unlike RBS and FTSP, Römer does not require a complex computation

mechanism based on message exchanges in the past, but only one previous message exchange.

Second best energy efficient method is RBS. RBS also takes into account post-facto synchronization. Clock drift is estimated by linear regression based line fitting on past clock phase offsets. Thus, computational complexity is considerably higher than Römer's and TSPN and requires more message exchanges than Römer's. However, it purveys energy savings by not updating local clocks of the nodes.

Energy utilization in TSPN is moderate. Although being a post-facto synchronization scheme, it has high computation and communication costs. In pairwise synchronization, the method requires three message exchanges between the nodes (time sync, synchronization pulse, and acknowledgment). This results in a higher communication load. Clock drift computation is not as complex as RBS or FTSP; but unlike RBS, local clocks of the nodes are updated.

FTSP is the most inefficient synchronization scheme among the other three, in terms of energy utilization. No argument on post-facto synchronization is made in the paper. Clock drift estimations are done in a similar way with RBS; hence, it has high computation cost and requires to have enough message exchanges to be completed before convergence. Also, the method updates the local clocks. All these result in high energy consumption for FTSP.

To sum up our conclusion for energy, three factors affect the efficiency of an algorithm. Among those three, post-facto synchronization plays the most important role. Then comes the computation complexity, including whether local times are updated or not. Finally, the communication complexity refers to the number of message exchanges required.

5.3. Scalability. Scalability requires a synchronization scheme to be able to scale well with size of the network. In other words, scalability measures how much the efficiency is affected by the increasing number of nodes. By 12 efficiencies, I mean precision and energy utilization and will be comparing the methods in terms of these two constraints.

Römer utilizes the energy in an efficient way. This makes it scalable in terms of energy consumption. However, accuracy of the system degrades by two things: age of time stamps and the number of hops. Even with a network of 5-hops, the average inaccuracy was reported as 3ms, which is a considerable low precision. Therefore, it can be concluded that scalability of Römer's method depends on the application. For a system, in which constraints on energy consumption are stricter than constraints on accuracy, Römer's would be the best scalable method. The method was tested on a prototype system; thus, scalability of the system was not discussed in the paper.

RBS is a scalable system in terms of both precision and energy efficiency. Although TSPN and FTSP achieve better precision than RBS, energy is utilized in a more efficient way by RBS. In their discussion, authors claim that TSPN is a very scalable system, considering that it would give better accuracy than RBS. However, TSPN has moderate scalability. This is because it requires a topological hierarchy and does not assume dynamic changes in the network structure. Also, the experiments reported by the TSPN paper use 300 nodes at the maximum case. This number is not enough, considering that number of sensor in a network can be in order of thousands.

FTSP is a scalable network in terms of accuracy. It would be the best system for an application, in which precision requirements are high and power is not an issue. Similar to TSPN, FTSP also exploits from a hierarchical network structure. However, unlike TSPN, the root node is selected dynamically among the nodes in the node. This makes FTSP more robust to failures in the networks. Experimental set of FTSP consists of 1000 motes, which is the highest number of nodes used among all other three systems, as well as many other synchronization schemes not mentioned here.

6. Conclusions

I have discussed synchronization issues in wireless sensor networks. Dynamic topological structure and limited capabilities of sensor nodes make the synchronization problem more difficult for sensor network environments. Moreover, possible delays in message exchange between the nodes make it even harder. There are certain issues to be considered in designing a synchronization scheme, such as multimodality, post-facto synchronization, single or multihop topology, and adaptability. The desired features of the synchronization scheme include energy efficiency, scalability, precision, lifetime, scope, and cost.

I have summarized four different synchronization schemes for wireless sensor networks. For comparison, I have analyzed three features—efficiency, scalability, and precision—of these systems in detail. It is seen that there is a trade-off between these features and no single system provides all these together.

Systems that provide high precision either remove possible sources of error on sender side by taking a receiver-receiver based approach or remove critical error sources by exploiting the MAC layer time stamping with a sender-receiver based approach. Furthermore, estimating the clock drift with linearregression on past clock offsets makes the system robust to possible failures in the network.

Post-facto synchronization, computation, and communication complexities determine the energy efficiency of a synchronization method. Systems that require synchronization when an event of interest occurs save energy by letting the nodes go idle and save power. Moreover, local clock updates also require high amount of energy. Systems, which do not require this, utilize the energy more efficiently.

Finally, scalability can be in terms of either precision or energy efficiency. It is seen that the more energy a system requires, the more precision that it can achieve and vice versa. So, scalability depends on application requirements, whether more precision or better energy utilization is needed.

References

[1] J. Wu, L. Jiao, and R. Ding, "Average time synchronization in wireless sensor networks by pairwise messages," *Computer Communications*, vol. 35, no. 2, pp. 221–233, 2012.

[2] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS—Operating Systems Review*, vol. 36, pp. 147–163, 2002.

[3] J. Elson and K. Romer, "Wireless sensor networks: a new regime for time synchronization," *ACM SIGCOMM Computer Communication Review*, vol. 33, pp. 149–154, 2003.

[4] M. K. Maggs, S. G. O'Keefe, and D. V. Thiel, "Consensus clock synchronization for wireless sensor networks," *IEEE Sensors Journal*, vol. 12, no. 6, pp. 2269–2277, 2012.

[5] J. Chen, Q. Yu, Y. Zhang, H.-H. Chen, and Y. Sun, "Feedback-based clock synchronization in wireless sensor networks: a control theoretic approach," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 6, pp. 2963–2973, 2010.

[6] S. Misra and A. Vaish, "Reputation-based role assignment for role-based access control in wireless sensor networks," *Computer Communications*, vol. 34, no. 3, pp. 281–294, 2011.

[7] B. Liu, F. Ren, J. Shen, and H. Chen, "Advanced self-correcting time synchronization in wireless sensor networks," *IEEE Communications Letters*, vol. 14, no. 4, pp. 309–311, 2010.

[8] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, no. 3, pp. 146–158, 1989.

[9] R. Gusella and S. Zatti, "The accuracy of the clock synchronization achieved by tempo in berkeley unix 4.3bsd," Tech. Rep. UCB/CSD-87-337, EECS Department, University of California, Berkeley, Calif, USA, 1987.

[10] C. Fidge, "Logical time in distributed computing systems," *Computer*, vol. 24, no. 8, pp. 28–33, 1991.

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

[12] D. L. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[13] F. Sivrikaya and B. Yener, "Time synchronization in sensor networks: a survey," *IEEE Network*, vol. 18, no. 4, pp. 45–50, 2004.

[14] K. Römer, "Time synchronization in ad hoc networks," in *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01)*, pp. 173–182, October 2001.

[15] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 3, no. 3, pp. 281–323, 2005.

[16] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[17] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," in *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, p. 186, San Francisco, Calif, USA, April 2001.

[18] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems (SenSys '03)*, pp. 138–149, November 2003.

[19] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems (SenSys '04)*, pp. 39–49, November 2004.