Available online at www.sciencedirect.com
SCIENCE DIRECT®
Ad Hoc Networks 3 (2005) 281–323
Ad Hoc Networks
www.elsevier.com/locate/adhoc

Clock synchronization for wireless sensor networks: a survey

Bharath Sundararaman, Ugo Buy *, Ajay D. Kshemkalyani

Department of Computer Science, University of Illinois at Chicago, 851 South Morgan Street, Chicago, IL 60607, United States

Received 17 September 2004; received in revised form 1 January 2005; accepted 18 January 2005

Abstract

Recent advances in micro-electromechanical (MEMS) technology have led to the development of small, low-cost, and low-power sensors. Wireless sensor networks (WSNs) are large-scale networks of such sensors, dedicated to observing and monitoring various aspects of the physical world. In such networks, data from each sensor is agglomerated using *data fusion* to form a single meaningful result, which makes time synchronization between sensors highly desirable. This paper surveys and evaluates existing clock synchronization protocols based on a palette of factors like precision, accuracy, cost, and complexity. The design considerations presented here can help developers either in choosing an existing synchronization protocol or in defining a new protocol that is best suited to the specific needs of a sensor-network application. Finally, the survey provides a valuable framework by which designers can compare new and existing synchronization protocols.

Keywords: Synchronization protocol; Sensor network; Wireless network; Comparative evaluation

1. Introduction

In recent years, tremendous technological advances have occurred in the development of low-cost sensors, which are capable of wireless communication and data processing [29,32,61,65]. Wireless sensor networks (WSNs) are distributed networks of such sensors, dedicated to closely observing real-world phenomena. Such sensors may be embedded in the environment or enabled with mobility; they can be deployed in inaccessible, dangerous, or hostile environments. The sensors need to configure themselves in a communication network, in order to collect information that has to be pieced together to assemble a broader picture of the environment than what each sensor individually senses. A huge surge of interest in this field has been triggered by applications in domains as diverse as military [16], environmental [7,8,63], medical [33], scientific [1,5,39,60,72,73],

* Corresponding author. Tel.: +1 312 413 2296; fax: +1 312 413 0024.
E-mail address: buy@cs.uic.edu (U. Buy).

1570-8705/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.adhoc.2005.01.002

Table 1
Example applications of wireless sensor networks

Domain	Applications
Military	Detection of nuclear, biological, and chemical attacks and presence of hazardous materials Prevention of enemy attacks via alerts when enemy aircrafts are spotted Monitoring friendly forces, equipment and ammunition
Environmental	Forest fire monitoring, flood detection, and earthquake detection Monitoring ecological and biological habitats
Civilian	Determining spot availability in a parking lot. Active badge tracking at the workplace Surveillance for security in banks and shopping malls. Highway traffic monitoring
Health	Tracking and monitoring doctors inside a hospital Identifying pre-defined symptoms by telemonitoring human physiological data
Home and K-12 education	The intelligent home and smart kindergarten, where wireless networks are used in developmental, problem-solving environments
Scientific	Space and interplanetary exploration. Deep undersea exploration Subatomic particle study. High-energy physics. Study of cosmic radiation

and industrial, civilian, and home networks [9,33,41,69,70], as shown in Table 1. As wireless sensor networks become an integral part of the modern era [6,13,14,22,23,26], addressing the issues in designing such networks becomes mandatory. Good sources of further information on wireless sensor networks and the challenges therein include surveys authored by Akyildiz et al. [2,3], by Culler and Hong [13], by Culler et al. [14], by Tilak et al. [64], and by Tubaishat and Madria [66].

In wireless sensor networks, the basic operation is *data fusion*, whereby data from each sensor is agglomerated to form a single meaningful result [17,44,71,74,76–78]. For instance, in forest fire monitoring, a forest fire can be detected by different sensors at different points in time when the fire enters the range of each sensor. Sensor readings (e.g., direction or velocity) and timestamps (indicating the time at which the fire was sensed) are passed along so that fusion of such information from various sensors will add up to a global result. In this case, it might be the time elapsed since the fire was first spotted and its direction. The fusion of individual sensor readings is possible only by exchanging messages that are timestamped by each sensor's local clock. This mandates the need for a *common notion of time* among the sensors. Protocols that provide such a common notion of time are *clock synchronization* protocols.

Researchers have developed successful clock synchronization protocols for wired networks over the past few decades. These are unsuitable for a wireless sensor environment because the challenges posed by wireless sensor networks are different and manifold. The most important differences are summarized here. First, wireless sensor networks can contain several thousands of sensors and their wide deployment is enabled by the fact that sensors are becoming cheaper and smaller in size. For example, a military tracking application may consist of hundreds of thousands of sensors, and scalability becomes a major issue. If a particular area needs sudden surveillance it might be populated with thousands of sensors in an ad-hoc manner, and the network must be scalable to the change. Second, self-configuration and robustness become a direct necessity in order to function under rapid deployment conditions and operation in inaccessible or dangerous environments. Third, energy conservation is a very important concern. It is impossible to provide a power source to each sensor in such a vast network, and the small sizes of sensors restrict the amount of energy that can be stored and procured.

This paper has three objectives. First, it presents a survey of clock synchronization protocols for the rapidly emerging wireless networks, based on a palette of factors such as precision, accuracy, cost, and complexity. The survey will help a reader

choose the most appropriate protocol for his application. Second, our analysis of issues underlying synchronization protocols will guide designers in defining new protocols tailored to specific applications of sensor networks. Finally, the survey establishes a framework for comparing new and existing clock synchronization protocols.

Although there are many surveys on wireless sensor networks, most of the existing surveys do not focus on time synchronization. Culler et al. recently published an overview of sensor networks in a special issue of *IEEE Computer* magazine [14]. Tilak et al. [64] present performance metrics for sensor networks along with a classification of sensor networks based on the organization of communicating nodes within a network. Their survey is especially valuable in its ability to relate the synchronization requirements to the organization of a sensor network. Akyildiz et al. wrote an excellent survey on sensor networks [3]. These authors discuss various communication protocols based on the layers of the Open System Interconnection (OSI) model. Several open research issues—including hardware design, modulation protocols, and strategies to overcome signal propagation effects—are discussed along with various MAC protocols. Hill et al. [29] discuss various hardware architectures for nodes in sensor networks. Their survey was published in a special issue of the *Communications of the ACM* on wireless sensor networks. The issue contains various other articles that discuss synchronization issues in these networks [13,63,71]. The work closest to ours is Elson and Romer's discussion of the design principles underlying synchronization protocols for wireless networks [20].

This paper is organized as follows. Section 2 discusses the foundations of clock synchronization and reviews traditional synchronization protocols for wired networks that are not constrained by the peculiar limitations of wireless sensor networks. It is necessary to understand these protocols to appreciate why new protocols had to be designed specifically for wireless sensor networks. Section 3 introduces the reader to clock synchronization in wireless sensor networks by explaining the peculiar characteristics of such networks that render the traditional synchronization protocols ineffective. The section next explains the design principles underlying clock synchronization protocols for wireless sensor networks. It then provides a classification of existing protocols based on synchronization issues and application-dependent features. Section 4 analyzes several existing clock synchronization protocols for wireless sensor networks. Section 5 presents a quantitative and qualitative comparison of clock synchronization protocols. The conclusions of our survey are given in Section 6.

2. Traditional clock synchronization

2.1. Motivation

In centralized systems, there is no need for synchronized time because there is no time ambiguity. A process gets the time by simply issuing a system call to the kernel. When another process then tries to get the time, it will get either an equal or a higher time value. Thus, there is a clear ordering of events and the times at which these events occur.

In distributed systems, there is no global clock or common memory. Each processor has its own internal clock and its own notion of time. In practice, these clocks can easily drift seconds per day, accumulating significant errors over time. Also, because different clocks tick at different rates, they may not remain always synchronized although they might be synchronized when they start. This clearly poses serious problems to applications that depend on a synchronized notion of time. For most applications and algorithms that run in a distributed system, we need to know time in one or more of the following aspects.

- The time of the day at which an event happened on a specific machine in the network.
- The time interval between two events that happened on different machines in the network.
- The relative ordering of events that happened on different machines in the network.

Unless the clocks in each machine have a common notion of time, time-based queries cannot be

answered. Some practical examples that stress the need for synchronization are listed below.

- In database systems, the order in which processes perform updates on a database is important to ensure a consistent, correct view of the database. To ensure the right ordering of events, a common notion of time between cooperating processes becomes imperative.
- Liskov [43] states that clock synchronization improves the performance of distributed algorithms by replacing communication with local computation. When a node N needs to query node M regarding a property, it can deduce the property with some previous information it has about node M and its knowledge of the local time in node M.
- It is quite common that distributed applications and network protocols use timeouts, and their performance depends on how well physically dispersed processors are time-synchronized. Design of such applications is simplified when clocks are synchronized.

Clock synchronization is the process of ensuring that physically distributed processors have a common notion of time. It has a significant effect on many areas like security systems, fault diagnosis and recovery, scheduled operations, database systems, and real-world clock values.

2.2. Clocks in distributed systems

In distributed systems where each machine has its own physical clock, we have seen that clock synchronization is of significant importance. Before delving into the details of synchronizing clocks, we define the notion of a clock. A *computer clock* is an electronic device that counts oscillations in an accurately-machined quartz crystal, at a particular frequency. It is also defined as an ensemble of hardware and software components used to provide an accurate, stable, and reliable time-of-day function to the operating system and its clients. Computer clocks are essentially timers. The timer counts the oscillations of the crystal, which is associated with a *counter register* and a *holding register*. For each oscillation in the crystal, the counter is decremented by one. When the counter becomes becomes zero, an interrupt is generated and the counter is reloaded from the holding register. Therefore, it is possible to program a timer to generate an interrupt 60 times a minute, where each interrupt is called a *clock tick*, by setting an appropriate value in the holding register. At each clock tick, the interrupt procedure increments the clock value stored in memory.

The clock value can be scaled to get the time of the day; the result can be used to timestamp an event on that computer. In practice, the quartz crystals in each of the machines in a distributed system will run at slightly different frequencies, causing the clock values to gradually diverge from each other. This divergence is formally called the *clock skew*, which can lead to an inconsistent notion of time. Clock synchronization is performed to correct this clock skew in distributed systems. There are two broad ways to achieve this.

- Clocks are synchronized to an accurate real-time standard like universal coordinated time (UTC). Clocks that must not only be synchronized with each other but also have to adhere to physical time are termed *physical clocks*. Such clocks are the subject of this paper.
- For applications in which causality-based logical time can be substituted for real-time (e.g., mutual exclusion requires only the logical condition that no two processes access the critical section concurrently), clocks are relatively synchronized to each other because the requirement is only to provide an ordering of events, and not the exact real-world time at which each event occurred. For clocks that provide only relative synchrony, only causality-based consistency of clocks matters as opposed to synchrony with respect to physical time. Such clocks are denoted by (causality-based) *logical clocks* [37,38]. Synchronizing such clocks will not be considered in this paper.

2.3. Clock inaccuracies

We require the following definitions. For any two clocks C_a and C_b, Fig. 1 gives our termino-

Time: The time of a clock in a machine p is given by the function $C_p(t)$, where $C_p(t) = t$ for a perfect clock.

Frequency : Frequency is the rate at which a clock progresses. The frequency at time t of clock C_a is $C_a^{'}(t)$.

Offset: Clock offset is the difference between the time reported by a clock and the *real time*. The offset of the clock C_a is given by $C_a(t) - t$. The offset of clock C_a relative to C_b at time $t \geq 0$ is given by $C_a(t) - C_b(t)$.

Skew: The skew of a clock is the difference in the frequencies of the clock and the perfect clock. The skew of a clock C_a relative to clock C_b at time t is $(C'_a(t) - C'_b(t))$.

If the skew is bounded by ρ, then as per Equation 1, clock values are allowed to diverge at a rate in the range of $1 - \rho$ to $1 + \rho$.

Drift (rate): The drift of clock C_a is the second derivative of the clock value with respect to time, namely $C''_a(t)$. The drift of clock C_a relative to clock C_b at time t is $(C''_a(t) - C''_b(t))$.

Fig. 1. Clock terminology.

logy, which is consistent with previous definitions [46,47,50].

The term *software clock* normally refers to the time in a computer clock to stress that it is just a counter that gets incremented for crystal oscillations. The interrupt handler must increment the software clock by one every time an interrupt (i.e., a clock tick) occurs. Most common clock hardware is not very accurate because the frequency that makes time increase is never exactly right. Even a frequency deviation of just 0.001% would cause a clock error of about one second per day. This is also a reason why clock performance is often measured with very fine units like one part per million (PPM).

Consider the physical clock synchronization of machines in a distributed system to UTC. At any point of time, if the time at UTC is t, the time in the clock of machine p is $C_p(t)$. In a perfect world, $C_p(t) = t$ for all p and all t. This means $dC/dt = 1$. However, due to the clock inaccuracy discussed above, a timer (clock) is said to be working within its specification if

$$1 - \rho \leqslant \frac{dC}{dt} \leqslant 1 + \rho, \tag{1}$$

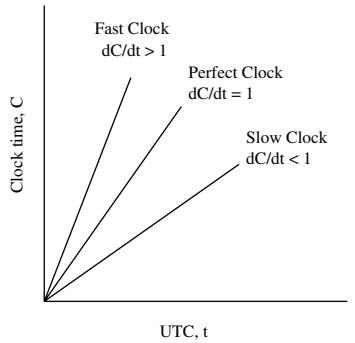

Fig. 2. Behavior of fast, slow, and perfect clocks with respect to UTC.

where constant ρ is the maximum skew rate specified by the manufacturer. Fig. 2 illustrates the behavior of fast, slow, and perfect clocks with respect to UTC.

2.4. Clock synchronization protocols

The requirements of a clock synchronization protocol are listed in Fig. 3. Numerous clock synchronization protocols have been developed in the

- The protocol should cope with unreliable network transmission and unbounded message latencies.
- When synchronizing two nodes, each node must be able to estimate the local time on the other node's clock. This is not a trivial issue due to non-deterministic message delays between the nodes in a distributed system.
- Time must never run backward. This implies that clocks must be gradually and gracefully advanced until the correction is achieved, rather than being outright set back.
- Synchronization overhead must not degrade system performance.

Fig. 3. Requirements of clock synchronization protocols.

past few decades. Some of the representative protocols are discussed next.

2.4.1. Remote clock reading method

Clock synchronization between any two nodes is generally accomplished by message exchanges, which allow one of the nodes to estimate the time in the other node's clock. Once the time difference between the clocks of the nodes is computed, the clocks can be corrected or adjusted to run in tandem. In the presence of non-deterministic and unbounded message delays, messages can get delayed arbitrarily, which makes synchronization very difficult. The effectiveness of a synchronization protocol centers around its ability to prevent non-deterministic message delays from affecting the quality of synchronization.

Cristian defined the *Remote Clock Reading* method, which handles unbounded message delays between processes [11]. This protocol is also used by other clock synchronizing methods [28,51]. When a process wants a time estimate of a remote process, it sends a time request and waits for the remote process to respond. When it receives the response, the process calculates the round-trip as the difference between the time at which it initiated the request and the time at which it received the response. The response contains the estimate of the time on the remote process. On obtaining such a response, it corrects its local clock to the sum of the estimate and half the round-trip time. Several trials are performed because the message delay is non-deterministic and the trial which offers the least round-trip time is chosen; alternately, the average of multiple trials is chosen. Cristian's method synchronizes several clients to an accurate time service (universal coordinated time) using the remote clock reading method, as shown in Fig. 4 and illustrated in Fig. 5.

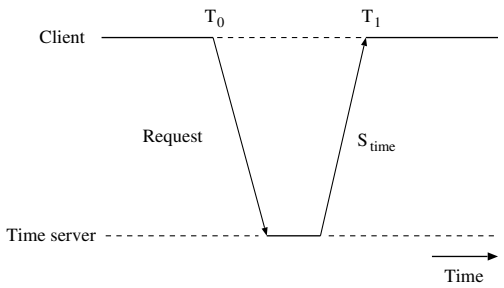

Fig. 5. Remote clock reading [11].

Drawbacks. A disadvantage of Cristian's protocol is that the time for any message to be sent is highly variable due to network traffic and message routing. These factors are not only hard to measure accurately but also unpredictable. This protocol also induces a high complexity in terms of the number of message exchanges, and there is no definitive means of deciding how many trials must be performed to reach an accurate round-trip time estimate.

2.4.2. Time transmission method

Arvind [4] defined the *Time Transmission Protocol* (*TTP*), which is used by a node to communicate the time on its clock to a target node. The target node then estimates the time in the source node by using the message timestamps and message delay statistics.

Algorithm. Assume that M is the source node and S is the target node. The algorithm for estimating the time is given in Fig. 6. Eq. (2) gives the specific formula to estimate the time.

- A client sends a message to the server requesting a timestamp. Let this message be initiated at time T_0 local to the client.
- The server then returns a message holding the timestamp (S_{time}). S_{time} is the local time at the server.
- The client receives this message at its local time, say, T_1.
- The client then sets its time to S_{time} (accurate time from the server) + (T_1-T_0)/2 (time required to transmit the message).
- To ensure accuracy, several round-trips are made and the average is used or the shortest round-trip is used.

Fig. 4. Cristian's synchronization protocol.

- M sends a series of synchronization messages to S. The i^{th} message is sent at time T_i of M's clock and received at time R_i of S's clock.
- S estimates M's time as,

$$T_{est} = R_n - (\bar{R}(n) - \bar{T}(n)) + \bar{d} \tag{2}$$

where,

$$\bar{R}(n) = \frac{1}{n}\sum_{i=1}^{n} R_i, \quad \bar{T}(n) = \frac{1}{n}\sum_{i=1}^{n} T_i \tag{3}$$

- $\bar{d}$ is the estimate of the expected value of message delay.
- R_n is the time at which the n^{th} message is received by S.

- T_{est} is the target's estimate of the time at the source. Once the time on the source is estimated, the target corrects its local clock to achieve synchronization.

Fig. 6. The time transmission algorithm [4].

Derivation. The derivation of formula (2) in Fig. 6 is as follows.

- The actual time T_{act} on the source node M's local clock is

$$T_{\text{act}} = R_n - \delta_n, \tag{4}$$

where δ_n is the offset between clocks S and M.
- Since δ_n is not determinable, the average value over n messages is

$$\bar{\delta}(n) = \frac{1}{n}\sum_{i=1}^{n} \delta_i. \tag{5}$$

- From this formulation, we get,

$$T_{\text{approx}} = R_n - \bar{\delta}(n). \tag{6}$$

- The ith message is received at S only d_i units after it is transmitted by M.

$$R_i = T_i + d_i + \delta_i. \tag{7}$$

It follows that,

$$\bar{\delta}(n) = \overline{R}(n) - \overline{T}(n) - \bar{d}(n). \tag{8}$$

Rearranging, we get

$$T_{\text{approx}} = R_n - \overline{R}(n) + \overline{T}(n) + \bar{d}(n). \tag{9}$$

As individual message delays are not known, $\bar{d}(n)$ is replaced with $\bar{d}$, an expected value of message delay. T_{approx} now becomes

$$T_{\text{est}} = R_n - \overline{R}(n) + \overline{T}(n) + \bar{d}. \tag{10}$$

Drawbacks. The protocol involves a large number of synchronization messages, resulting in a high computational overhead. The number of messages required for synchronization reduces the applicability of the protocol.

2.4.3. Offset delay estimation method

The *offset delay estimation* method is employed by the *Network Time Protocol* (*NTP*) [48] which is widely used for clock synchronization on the Internet. The design of NTP involves a hierarchical tree of time servers. The primary server at the root synchronizes with the UTC. The next level contains secondary servers, which act as a backup to the primary server. At the lowest level is the synchronization subnet which has the clients.

Clock offset and delay estimation. In practice, a source node cannot accurately estimate the local time on the target node due to varying message or network delays between the nodes. This protocol employs a very common practice of performing several trials and chooses the trial with the minimum delay. Recall that Cristian's remote clock reading method [11] also relied on the same strategy to estimate message delay.

Fig. 7 shows how NTP timestamps are numbered and exchanged between peers A and B. Let T_1, T_2, T_3, T_4 be the values of the four most recent timestamps as shown. Assume that clocks A and B are stable and running at the same speed. Let $a = T_1 - T_3$ and $b = T_2 - T_4$. If the network delay difference from A to B and from B to A, called

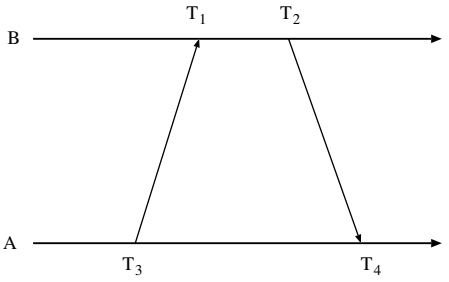

Fig. 7. Offset and delay estimation [48].

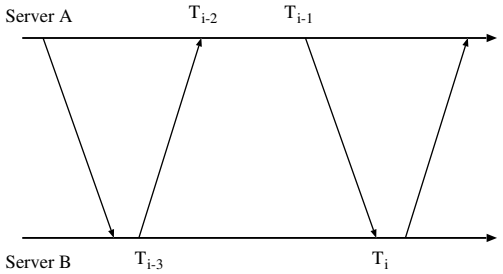

Fig. 8. Timing diagram for the two servers [48].

differential delay, is small, the clock offset θ and round-trip delay δ of B relative to A at time T_4 are approximately given by the following.

$$\theta = \frac{a+b}{2}, \quad \delta = a - b. \tag{11}$$

Each NTP message includes the latest three timestamps T_1, T_2 and T_3, while T_4 is determined upon arrival. Thus, both peers A and B can independently calculate delay and offset using a single bidirectional message stream as shown in Fig. 8. The NTP protocol is shown in Fig. 9.

Drawbacks. The offset delay estimation protocol is similar to Cristian's method [11] due to its averaging approach. The disadvantage of both methods is that they lead to a high synchronization overhead in terms of message complexity and reduced accuracy. However, accuracy of the network time protocol is better than for Cristian's protocol because delays are partly compensated.

2.4.4. Set-valued estimation method

The set-valued estimation method [40] is particularly useful in systems where modeling uncer-

- A pair of servers in symmetric mode exchange pairs of timing messages.
- A store of data is then built up about the relationship between the two servers (pairs of offset and delay).
  Specifically, assume that each peer maintains pairs (O_i, D_i), where
  O_i - measure of offset (θ)
  D_i - transmission delay of two messages (δ).
- The offset corresponding to the minimum delay is chosen.
  Specifically, the delay and offset are calculated as follows. Assume that message m takes time t to transfer and m' takes t' to transfer.
  - The offset between A's clock and B's clock is O. If A's local clock time is $A(t)$ and B's local clock time is $B(t)$, we have
    $$A(t) = B(t) + O \tag{12}$$
    Then,
    $$T_{i-2} = T_{i-3} + t + O \tag{13}$$
    $$T_i = T_{i-1} - O + t' \tag{14}$$
    Assuming $t = t'$, the offset O_i can be estimated as:
    $$O_i = (T_{i-2} - T_{i-3} + T_{i-1} - T_i)/2 \tag{15}$$
    The round-trip delay is estimated as:
    $$D_i = (T_i - T_{i-3}) - (T_{i-1} - T_{i-2}) \tag{16}$$
  - The eight most recent pairs of (O_i, D_i) are retained.
  - The value of O_i that corresponds to minimum D_i is chosen to estimate O.

Fig. 9. The network time protocol synchronization protocol [48].

- Processor P_i sends out N messages to processor P_j at local times $\underline{t_{ik}}$ for $k = 1, \ldots, N$.
- Processor P_i receives replies to these messages from processor P_j at times $\overline{t_{ik}}$ and each received message is stamped with $\underline{t_{ik}}$ and the local time, t_{jk}, when processor P_j received the k^{th} message. The message-passing events between P_i and P_j are shown in the space-time diagram in Figure 11.
- When the last reply is received from processor P_j, the originating processor P_i has a set of time-stamped triples,

$$(\underline{t_{ik}}, t_{jk}, \overline{t_{ik}}) \tag{18}$$

Using the triples, a graph is plotted with the local time on processor P_j on the X-axis and local time on processor P_i on the Y-axis. Each data triple can be plotted as an error bar, as shown in Figure 12. The relative drift a_{ij} and the relative offset b_{ij} are determined from the slope and Y-intercept of any line that passes through all of the error bars.

Fig. 10. The set-valued estimation protocol, shown for a pair of processes P_i and P_j [40].

tainty is not directly captured by a priori models. Assume that a distributed system consists of processors P_i, for $i = 1, \ldots, N$. Let t_i denote the local time on the clock for processor P_i. We assume that the local times t_i and t_j on processors P_i and P_j, respectively, can be related by the linear equation:

$$t_i = a_{ij} t_j + b_{ij}, \tag{17}$$

where a_{ij} and b_{ij} represent the relative skew and offset between the two hardware clocks.

For any two processors P_i and P_j which pass messages between each other, the protocol works as shown in Fig. 10.

Clock correction. Clock correction is performed using an algorithm by Dolev et al. [18], which works as follows. Synchronization proceeds in rounds and the interval between synchronization rounds is predefined. The node that first reaches the end of the synchronization round is obviously the fastest and all other nodes have to adjust their rate to match the rate of this node.

There are two tasks associated with each node, *time monitor* and *message handler*. If a node reaches the end of the synchronization period before receiving any message from other nodes, it performs the time monitor task by sending a message to every other node indicating that it has reached the end of its synchronization period first. Every other node will execute the message handler to process the incoming message and adjust its clock to the fastest clock.

Drawbacks. The advantage of this protocol is that every node tries to act as a synchronizer at the same time and at least one succeeds. There is no single point of failure associated with the system. However, message corruption and message losses might greatly degrade the accuracy. In addition, there is a possibility of instability during clock correction because nodes always seek to adjust their clock rates to match the fastest clock in the

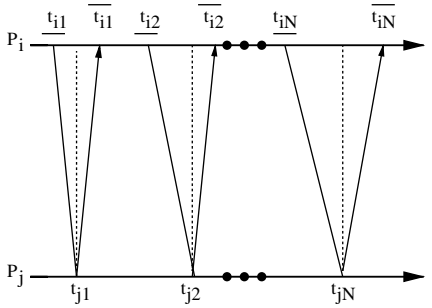

Fig. 11. Message passing between P_i and P_j [40].

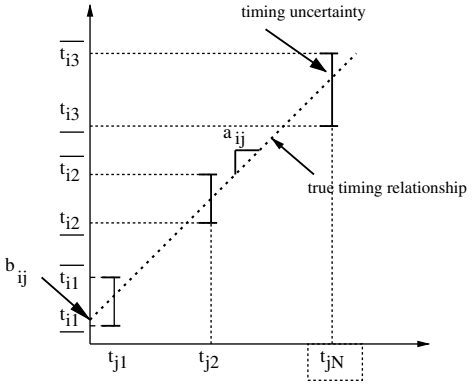

Fig. 12. Data triples plotted with the local time of P_j on the X-axis and the local time of P_i on the Y-axis [40].

network. If a processor P_i overcorrects its clock (e.g., because of variations in message transmission delays), this will trigger a new round of corrections as all other nodes will try to match the new clock rate of P_i. Worse yet, this phenomenon may repeat itself and continue indefinitely, leading to faster and faster clock rates throughout the network.

3. Clock synchronization in wireless sensor networks

The traditional clock synchronization protocols surveyed in the previous section are widely used in wired networks. However, they are not suitable for wireless sensor networks for a variety of reasons that we discuss in this section. Clock synchronization in wireless sensor networks requires newer and more robust approaches. A thorough understanding of the challenges posed by wireless sensor networks is crucial for the successful design of synchronization protocols for such networks. This section examines the design principles for clock synchronization in wireless sensor networks, and then classifies various such synchronization protocols.

3.1. Challenges of sensor networking

Wireless sensor networks have tremendous potential because they will expand our ability to monitor and interact remotely with the physical world. Smart sensors have the ability to collect vast amounts of hitherto unknown data, which will pave the way for a new breed of computing applications as we showed in Table 1. Sensors can be accessed remotely and placed where it is impractical to deploy data and power lines. Nodes can be spaced closely, yielding fine-grained pictures of real-world phenomena that are currently modeled only on a large scale. However, to exploit the full potential of sensor networks, we must first address the peculiar limitations of these networks and the resulting technical issues. Evidently, sensor networks can be best exploited by applications that perform data fusion to synthesize global knowledge from raw data on the fly. Although data fusion requires that nodes be synchronized, the synchronization protocols for sensor networks must address the following features of these networks.

3.1.1. Limited energy

While the efficiency of computing devices is increasing rapidly, the energy consumption of a wireless sensor network is becoming a bottleneck. Due to the small size and cheap availability of the sensors, sensor networks can employ thousands of sensors. This makes it impossible to wire each of these sensors to a power source. Also, the need for unmanned operation dictates that the sensors be battery-powered. Since the amount of energy available to such sensors is quite modest, synchronization must be achieved while preserving energy to utilize these sensors in an efficient fashion.

3.1.2. Limited bandwidth

In wireless sensor nets, much less power is consumed in processing data than transmitting it. Presently, wireless communication is restricted to a data rate in the order of 10–100 Kbits/s [21]. Pottie and Kaiser [55] have shown that the energy required to transmit 1 bit over 100 m, which is 3 joules, can be used to execute 3 million instructions. Bandwidth limitation directly affects message exchanges among sensors, and synchronization is impossible without message exchanges.

3.1.3. Limited hardware

The hardware of a sensor node is usually very restricted due to its small size. A typical sensor node like the Berkeley Mica2 mote [35] has a small solar battery, an 8-bit CPU that runs at a speed of 10 MHz, 128 KB to 1 MB memory, and a communication range of less than 50 m. Hill et al. [29] surveyed some sensor-network platforms as well as the most popular sensor architectures, such as Spec, Smartdust, Intel's Imote [32], and Stargate. Fig. 13 illustrates the configuration of a typical sensor

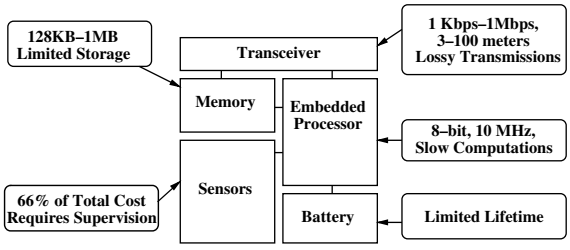

Fig. 13. Sensor node hardware for Mica mote [29].

node. The restrictions on computational power and storage space pose a huge challenge. The size of a sensor cannot be increased because it would make it more expensive and consume more power. This would prevent the deployment of thousands of sensor nodes, which is usually required for efficient operation of several critical applications.

3.1.4. Unstable network connections

An implicit advantage usually available to a wireless network is mobility. Mobile ad-hoc networks are becoming increasingly popular and the following issues must be addressed.

- The communication range of the mobile sensors is very limited (roughly 20–100 m), which makes message exchanges between sensor nodes difficult.
- A wireless medium is unshielded to external interference and this may lead to a high percentage of message loss.
- A wireless connection suffers from a restricted bandwidth and intermittent connectivity.
- The network topology frequently changes due to the mobility of the nodes. Dynamic reconfiguration becomes necessary.

3.1.5. Tight coupling between sensors and physical world

WSNs seek to monitor real-world phenomena and the network design is tailored to the specific environment being sensed. Therefore, as WSNs are used for critical and diverse applications like military tracking, forest fire monitoring, and geographical surveillance, the network has to be tailored to suit the application. For instance, sensors can be used to measure temperature, light, sound, or humidity, and the application (e.g., forest fire monitoring) decides the type of sensors to be used (e.g., temperature sensors).

3.2. Design principles of clock synchronization in sensor networks

Researchers have developed a wide variety of clock synchronization protocols for traditional wired networks over the past few decades, as surveyed in Section 2. However, due to the peculiar characteristics, limitations, and the dynamic nature of wireless sensor networks, as seen in Section 3.1, these protocols cannot be applied directly. Several important design considerations are listed next.

3.2.1. Energy efficiency

- External time standard (GPS) usage
  In sensor nets, energy conservation is very important. Traditional protocols like NTP [48] and TEMPO [28] use an external standard like Global Positioning System (GPS) or Universal Time (UTC) to synchronize the network to an accurate time source. However, the use of GPS poses a high demand for energy which is usually not available in sensor networks. This makes it difficult to maintain a common notion of time.
- Mode of transmission
  Reduction of energy is achieved by choosing to transmit over multiple short distances instead of a single long path. This translates into either a lower transmit power or a higher data transmission speed over a given distance. Either one will decrease the total end-to-end energy needed to transmit a packet of data. This implies that in large sensor networks, data is transmitted in sequences or hops, instead of a single long path from the sender to the receiver.
- Proactive versus reactive routing
  A proactive protocol keeps track of all the nodes in a node's neighborhood, having total knowledge of all possible routes at all times. Reactive protocols do not maintain routing information proactively and find routes only when they need them. A reactive protocol leads to energy savings because nodes do not waste energy by attempting to maintain synchronization at all times. Nodes are awakened only when they are needed. Elson et al.'s Reference Broadcast Synchronization (RBS) [19] uses a similar technique, called *post-facto synchronization*.

3.2.2. Infrastructure

In many critical sensor applications, the network is deployed in an ad-hoc fashion. Ad-hoc networks are networks of mobile wireless sensors

in which the mobile nodes constantly change their neighborhood and the configuration. This denies the convenience of having an infrastructure like NTP [48], which has several layers of servers that provide an accurate source of time. In ad-hoc sensor networks, the nodes must cooperate to organize themselves into a network and resolve contention for the available bandwidth. These tasks become more complex if the number of nodes grows or if the relationship among nodes changes rapidly, for instance, because of mobility.

3.2.3. End-to-end latency

Traditional wired networks are fully connected networks in which the variability in the propagation and (intermediate) queuing delay is relatively small. In addition, any node can send a message directly to another node at any point in time. This implies a constant end-to-end delay throughout the network and provides a close approximation for the actual latency. Sensor nets may be large in size and have to deal both with mobility and wireless transmission over a shared medium. These features make it impractical to assume a single latency bound between the ends of the network. Sensor nets therefore need localization algorithms to reduce this *latency error* as well as the *jitter*, the unpredictable variation in transmission times. Also, protocols that assume a fully connected network cannot be applied to multihop sensor networks.

3.2.4. Message loss and message delivery

Fault-tolerant algorithms for traditional wired networks handle message loss by sending extra messages. This ensures that every node participates in the synchronization, leading to better operation. Several protocols for wired networks employ the averaging method to compute the delay between two nodes, which is a critical aspect in maintaining synchronized time. Message loss handling and estimating message delay by averaging are not desirable in sensor nets because of the following reasons.

- Transmission of every bit requires energy and multiple message transmissions to estimate average delays lead to higher energy requirements.
- Message delivery is very unreliable due to the dynamic nature of the network, the intermittent connectivity, and the limited communication range of each node.

3.2.5. Network dynamics

A stationary sensor network, without any mobility, usually requires an initial set-up before beginning operation. However, if the application demands a higher population of nodes in a particular part of the network, the addition of extra nodes changes the neighborhood of each node and the configuration of the network. Dynamic sensor networks add further challenges because the nodes are mobile. Mobility directly leads to a frequent change in topology of the network. Hence, the protocols used for such networks, whether stationary or dynamic, must ensure self-configuration (by use of suitable neighborhood definition or leader election protocols) to achieve synchronization.

3.3. Classification of synchronization protocols

Wireless sensor networking can be applied to a wide range of applications, from simple parking lot monitoring to safety-critical applications like earthquake detection. As most networks are very closely coupled to the application, the protocols used for synchronization differ from each other in some aspects and resemble each other in other aspects. We classify synchronization protocols based on two kinds of features.

1. Synchronization issues
2. Application-dependent features

3.3.1. Synchronization issues

Wireless sensor networks provide answers to user queries by fusing data from each sensor to form a single answer or result. To accomplish this *data fusion*, it becomes necessary for these sensors to agree on a common notion of time. All the participating sensors can be enveloped in a common timescale by either synchronizing the local clocks in each sensor or by just translating timestamps

that arrive at a sensor into the local clock times. Various options are now described.

- *Master–slave versus peer-to-peer synchronization*
  Master–slave. A master–slave protocol assigns one node as the master and the other nodes as slaves. The slave nodes consider the local clock reading of the master as the reference time and attempt to synchronize with the master. In general, the master node requires CPU resources proportional to the number of slaves, and nodes with powerful processors or lighter loads are assigned to be the master node. Mock et al. [49] have adopted the IEEE 802.11 clock synchronization protocol due to its simple, non-redundant, master/slave structure. Ping's protocol [54] also adheres to the master–slave mode.
  Peer-to-peer. Most protocols in the literature, such as RBS [19], Romer's protocol [56], the protocol by PalChaudhuri et al. [53], the time diffusion protocol by Su and Akyildiz [62], and the asynchronous diffusion protocol of Li and Rus [42] are based on a peer-to-peer structure. Any node can communicate directly with every other node in the network. This eliminates the risk of master node failure, which would prevent further synchronization. Peer-to-peer configurations offer more flexibility but they are also more difficult to control.
- *Clock correction versus untethered clocks*
  Clock correction. Most methods in practice perform synchronization by correcting the local clock in each node to run on par with a global timescale or an atomic clock, which is used to provide a convenient reference time. The protocol of Mock et al. [49] and Ping's protocol [54] are based on this method. The local clocks of nodes that participate in the network are corrected either instantaneously or continually to keep the entire network synchronized.
  Untethered clocks. Achieving a common notion of time without synchronization is becoming popular, because a considerable amount of energy can be saved by this approach. RBS [19] builds a table of parameters that relate the local clock of each node to the local clock of every other node in the network. Local timestamps are then compared using the table. In this way, a global timescale is maintained while letting the clocks run untethered. Romer [56] uses the same principle. When timestamps are exchanged between nodes, they are transformed to the local clock values of the receiving node. The round-trip delay between two nodes and the idle time of a message are taken into consideration.
- *Internal synchronization versus external synchronization*
  Internal synchronization. In this approach, a global time base, called *real-time*, is not available from within the system and the goal is to minimize the maximum difference between the readings of local clocks of the sensors. The protocol of Mock et al. [49] uses internal synchronization.
  External synchronization. In external synchronization, a standard source of time such as Universal Time (UTC) is provided. Here, we do not need a global time base since we have an atomic clock that provides actual real-world time, usually called *reference time*. The local clocks of sensors seek to adjust to this reference time in order to be synchronized. Protocols like NTP [48] synchronize in this fashion because external synchronization is better suited to loosely coupled networks like the Internet. Most protocols in sensor networks do not perform external synchronization unless the application demands it, because energy efficiency is a primary concern and employing an external time source typically induces high-energy requirements.

  Internal synchronization usually leads to a more correct operation of the system, while external synchronization is primarily used to give users convenient reference time information. Note that internal synchronization can be performed in a master–slave or peer-to-peer fashion. External synchronization cannot be performed in a peer-to-peer fashion; it requires a master node which communicates with a time service like GPS to synchronize the slaves and itself to the reference time.

- *Probabilistic versus deterministic synchronization*
  Probabilistic synchronization. This technique provides a probabilistic guarantee on the maximum clock offset with a failure probability that can be bounded or determined. The reasoning behind a probabilistic approach is that a deterministic approach usually forces the synchronization protocol to perform more message transfers and induces extra processing. In a wireless environment where energy is scarce, this can be very expensive. The protocol of Pal-Chaudhuri et al. [53] is a probabilistic variation of RBS [19]. Arvind [4] defined a probabilistic protocol for wired networks.
  Deterministic synchronization. Arvind [4] defines deterministic algorithms as those that guarantee an upper bound on the clock offset with certainty. Most algorithms in the literature are deterministic. Sichitiu and Veerarittiphan's protocol [58] is centered on a deterministic algorithm. RBS [19] and the time-diffusion protocol [62] are also deterministic.
- *Sender-to-receiver versus receiver-to-receiver synchronization*
  Most existing methods synchronize a sender with a receiver by transmitting the current clock values as timestamps. As a consequence, these methods are vulnerable to variance in message delay. Newer methods such as RBS [19] perform synchronization among receivers using the time at which each receiver receives the same message. Such an approach reduces the *time-critical path*, which is the path of a message that contributes to non-deterministic errors in the protocol.
  Sender-to-receiver synchronization. This traditional approach usually happens in threesteps.

  1. The sender node periodically sends a message with its local time as a timestamp to the receiver.
  2. The receiver then synchronizes with the sender using the timestamp it receives from the sender.
  3. The message delay between the sender and receiver is calculated by measuring the total round-trip time, from the time a receiver requests a timestamp until the time it actually receives a response.

  The drawbacks of this approach are obvious. There is a variance in message delay between the sender and the receiver. The variance is due to network delays (prominent in multi-hop networks) and the workload in the nodes that are involved. Most methods compute the average message delay after performing many trials, during which they lose accuracy and add further overhead. Also, optimization of the time taken by the sender to prepare and transmit the message, and the time taken by the receiver to process the message must be considered.
  Receiver-to-receiver synchronization. This approach exploits the property of the physical broadcast medium that if any two receivers receive the same message in single-hop transmission (see below), they receive it at approximately the same time. Instead of interacting with a sender, receivers exchange the time at which they received the same message and compute their offset based on the difference in reception times. The obvious advantage is the reduction of the message-delay variance. This protocol is vulnerable only to the propagation delay to the various receivers and the differences in receive time.

Table 2 classifies the various protocols for clock synchronization, based on the analysis in this section.

3.3.2. Application-dependent features

- *Single-hop versus multi-hop networks*
  Single-hop communication. In a single-hop network, a sensor node can directly communicate and exchange messages with any other sensor in the network. However, many wireless sensor-network applications span several domains or neighborhoods. (Nodes within a neighborhood can communicate via single-hop message transmission.) The network is often too large, making it impossible for each sensor node to directly exchange messages with every other node. Elson and Romer [20] show that a single

Table 2
Classification based on synchronization issues

Protocol	Synchronization issues				
	Master–slave vs. peer-to-peer	Internal vs. external	Probabilistic vs. deterministic	Sender-to-receiver vs. receiver-to-receiver	Clock correction
RBS [19]	Peer-to-peer	Both	Deterministic	Receiver-to-receiver	No
Romer [56]	Peer-to-peer	Internal	Deterministic	Sender-to-receiver	No
Mock et al. [49]	Master/slave	Internal	Deterministic	Receiver-to-receiver	Yes
Ganeriwal et al. [25]	Master/slave	Both	Deterministic	Sender-to-receiver	Yes
Ping [54]	Master/slave	Both	Deterministic	Sender-to-receiver	Yes
PalChaudhuri et al. [53]	Peer-to-peer	Both	Probabilistic	Receiver-to-receiver	No
Sichitiu and Veerarittiphan [58]	Peer-to-peer	Internal	Deterministic	Sender-to-receiver	Yes
Time-diffusion protocol [62]	Peer-to-peer	Internal	Deterministic	Receiver-to-receiver	Yes
Asynchronous diffusion [42]	Peer-to-peer	Internal	Deterministic	Sender-to-receiver	Yes

latency bound cannot be assumed. Protocols such as those by Mock et al. [49], Ganeriwal et al. [24,25], Ping [54], and PalChaudhuri et al. [53] are based on single-hop communication; however, they can be extended to multi-hop communication.

Multi-hop communication. The need for multi-hop communication arises due to the increase in the size of wireless sensor networks. In such settings, sensors in one domain communicate with sensors in another domain via an intermediate sensor that can relate to both domains [19]. Communication can also occur as a sequence of hops through a chain of pairwise-adjacent sensors. RBS [19], Ping's protocol [54], the protocol by PalChaudhuri et al. [53], and Su and Akyildiz's time-diffusion protocol [62] can be suitably extended to handle multi-hop communication.

- *Stationary networks versus mobile networks*
  Most sensor networks are tightly coupled to the application and synchronization protocols vary depending on the application at hand. Mobility is an inherent advantage of a wireless environment but it induces more difficulties in achieving synchronization. It leads to frequent changes in network topology and demands that the protocol be more robust.
  Stationary networks. In stationary sensor networks, the sensors do not move. An example is a network of sensors for monitoring the motion of a vehicle in a certain area. For these sensor networks, the topology remains unchanged once the sensors are deployed in the environment. The protocols used by RBS [19], Mock et al. [49], Ganeriwal et al. [24,25], and PalChaudhuri et al. [53] are geared to stationary networks.
  Mobile networks. In a mobile network, the sensors have the ability to move, and they connect with other sensors only when entering the geographical scope of those sensors. The scope of a mobile sensor is the communication range up to which it can communicate and successfully exchange messages with other sensors. Romer [56] shows the need for a robust protocol, which can handle the frequent changes in network topology due to the mobility of the nodes. The change in topology is often a problem because it requires resynchronization of nodes and recomputation of the neighborhoods or clusters.
- *MAC-layer-based approach versus standard approach*
  The Media Access Control (MAC) layer is a part of the Data Link Layer of the Open System Interconnection (OSI) model. This layer is responsible for the following functions.
  – Providing reliability to the layers above it with respect to the connections established by the physical layer.
  – Preventing transmission collisions so that the message transmission between one sender and the intended receiver node(s) does not interfere with transmission by other nodes.

  MAC protocols effectively utilize the MAC layer to achieve better energy efficiency, which is crucial in sensor networks. The IEEE 802.11 MAC

protocol [31] is widely used. Several variations of this protocol have been defined for the purpose of controlling power consumption, including the protocols listed below. A survey by Chen et al. [10] compares some of these protocols.

- Sensor-MAC (S-MAC) [75],
- Power-Aware Multi-Access Protocol (PAMAS) [59],
- Energy-Conserving MAC (EC-MAC) [34],
- Packet-Reservation Multiple Access MAC (PRMA) [27],
- Distributed-Queuing Request Update Multiple Access (DQRUMA) [36],
- Multiservice Dynamic Reservation TDMA (MDR-TDMA) [52].

Reference Broadcast Synchronization [19] does not rely on MAC protocols in order to avoid a tight integration of the application with the MAC layer. The protocols used by Mock et al. [49], Ganeriwal et al. [24,25], and Sichitiu and Veerarittiphan [58] rely on the CSMA/CA protocol for the MAC layer. A survey of MAC protocols for sensor networks is given by Jones et al. [34].

Table 3 classifies the various protocols for clock synchronization, based on the analysis in this section.

4. Discussion of synchronization protocols

In the previous section, we analyzed the issues underlying synchronization in sensor networks. Now we summarize various existing synchronization protocols and their relationships to the above issues. We also discuss the relative advantages and disadvantages of these protocols. Given that sensor networks are generally closely tied to the real-world environment that they monitor, different networks will have different characteristics affecting their synchronization requirements. For this reason, some of the protocols that we discuss below will be more suitable than others in some cases and vice versa.

We will specifically consider the following protocols.

1. *Reference Broadcast Synchronization* (*RBS*) seeks to reduce non-deterministic latency using receiver-to-receiver synchronization and to conserve energy via post-facto synchronization [19].
2. *Romer's protocol*, which has been successfully applied to mobile ad-hoc networks, uses an innovative time transformation algorithm for achieving clock synchronization [56]. This protocol appears to be especially effective in environments with strict resource constraints.
3. *Mock's protocol* extends the IEEE 802.11 master–slave protocol by exploiting the tightness property of the communication medium [49]. Minimal message complexity and fault tolerance are the main benefits of this protocol.
4. *Network-wide Time Synchronization* is geared toward networks with a large node density [24].
5. *Delay Measurement Time Synchronization Protocol* for wireless sensor networks [54] is an energy-efficient protocol due to its low message

Table 3
Classification based on application-dependent features

Protocol	Application-dependent features		
	Single-hop vs. multi-hop	MAC layer vs. standard	Geared to mobility
RBS [19]	Both	Standard	No
Romer [56]	Both	Standard	Yes
Mock et al. [49]	Single-hop	MAC layer	No
Ganeriwal et al. [25]	Both	MAC Layer	Yes
Ping [54]	Both	Standard	No
PalChaudhuri et al. [53]	Both	Standard	No
Sichitiu and Veerarittiphan [58]	Both	MAC Layer	No
Time-diffusion protocol [62]	Both	Standard	Yes
Asynchronous diffusion [42]	Both	Standard	Yes

complexity. It is also lighter in computational cost, albeit less accurate, than the RBS protocol [19].

6. The *Probabilistic Clock Synchronization Service* for sensor networks extends RBS by providing probabilistic bounds on the accuracy of clock synchronization [53].
7. *Sichitiu and Veerarittiphan's protocol* provides good synchronization accuracy in wireless sensor networks while using a deterministic protocol with minimal computational and storage complexity [58].
8. The *Time-Diffusion Protocol* (TDP) by Su and Akyildiz achieves a network-wide "equilibrium" time using an iterative, weighted averaging technique based on a diffusion of messages involving all the nodes in the synchronization process [62].
9. The *Asynchronous Diffusion* protocol by Li and Rus uses a strategy similar to TDP; however, network nodes execute the protocol and correct their clocks asynchronously with respect to each other [42].

4.1. Reference broadcast synchronization [19]

The *Reference Broadcast Synchronization* (RBS) protocol is so named because it exploits the broadcast property of the wireless communication medium [19]. According to this property, two receivers located within listening distance of the same sender will receive the same message at approximately the same time. In other words, a message that is broadcast at the physical layer will arrive at a set of receivers with very little variability in its delay. If each receiver records the local time as soon as the message arrives, all receivers can synchronize with a high degree of precision by comparing their local clock values when the message was received. This protocol uses a sequence of synchronization messages from a given sender in order to estimate both offset and skew of the local clocks relative to each other. The protocol exploits the concept of *time-critical path*, that is, the path of a message that contributes to non-deterministic errors in a protocol. Fig. 14 compares the time-critical path of traditional protocols, which are based on sender-to-receiver synchronization, with receiver-to-receiver synchronization in RBS.

Non-deterministic transmission delays are detrimental to the accuracy of a synchronization protocol because they make it difficult for a receiver to estimate the time at which a message was sent and vice versa. In general, the time involved in sending a message from a sender to a receiver is the result of the following four factors, all of which can vary non-deterministically.

1. Send time: The time spent by the sender for message construction and the time spent to transmit the message from the sender's host to the network interface.
2. Access time: The time spent waiting to access the transmit channel.
3. Propagation time: The time taken for the message to reach the receiver, once it has left the sender.

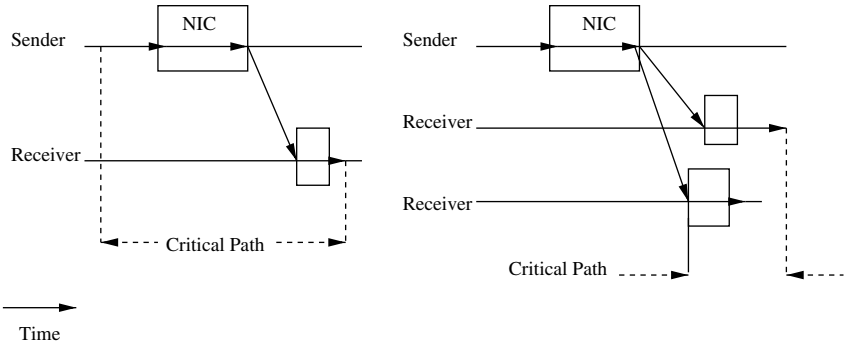

Fig. 14. Time-critical path for traditional protocols (left) and RBS protocol (right) [19].

4. Receive time: The time spent by the receiver to process the message.

By considering only the times at which a message reaches different receivers, the RBS protocol directly removes two of the largest sources of non-determinism involved in message transmission, namely the send time and the access time. Thus, this protocol can provide a high degree of synchronization accuracy in sensor networks. The RBS protocol uses the algorithm shown in Fig. 15 to estimate the phase offset between the clocks of two receivers.

This protocol can produce highly accurate results if message reception by each receiver is tight and if each receiver can record its local clock reading as soon as the message is received. This is often the case for single-hop communication in a wireless network. In practice, however, messages sent over a wireless sensor network can be corrupted. In addition, a receiving node may not be able to record the time of message arrival promptly, for instance, if the node was busy with other computations when the message arrived. To alleviate these non-deterministic factors, the RBS protocol uses a sequence of reference messages from the same sender, rather than a single message. Receiver j will compute its offset relative to any other receiver i as the average of clock differences for each packet received by nodes i and j:

$$\text{Offset}[i,j] = \frac{1}{m}\sum_{k=1}^{m}(T_{i,k} - T_{j,k}). \tag{19}$$

In this equation, parameters i and j denote two receivers, m is the number of reference broadcasts, and $T_{i,k}$ is node i's clock when it receives broadcast k.

Elson et al. [19] conducted an experiment with an actual network of n sensor nodes that were given random clock offsets; the times of m message transmissions were selected randomly. Each synchronization message was delivered to every receiver and timestamped using the receiver's clock. Since every receiver computes its offset with every other receiver, $O(n^2)$ offsets were obtained and compared with the actual offsets. The maximum difference between computed and actual offsets was considered to be the group dispersion.

In order to show that the precision of offset estimation increased with the number of broadcasts, an experiment was conducted for values of m between 1 and 50 broadcasts and values of n between 1 and 20 receivers for a total of 1000 trials. In the case of two receivers, the results show that when 30 reference broadcasts were sent instead of one broadcast, the precision improved from an error of 11 μs to 1.6 μs. A precision of the order of microseconds is clearly an excellent accuracy result for a sensor network.

The RBS protocol also estimates the skew between clocks in neighboring nodes of a sensor network. The skew computation must use multiple reference broadcasts in order to observe variations in the offset of two node clocks over time. Elson et al. [19] use the least squares method to find a best-fit line that will provide an estimate of another node's clock skew.

An experiment was conducted with a network of Berkeley motes [35]. A reference packet with a sequence number was periodically broadcast in a network of five motes. Each mote used a 2 μs resolution clock to timestamp the reception times of incoming broadcasts.

Fig. 16 shows a diagram by Elson et al. in which they plot the performance of their protocol [19]. Each point in the diagram represents the difference between the times at which the two nodes reported receiving a reference broadcast, plotted on a time-

1. A transmitter broadcasts a reference packet to two receivers.
2. Each receiver records the time at which the packet was received, according to its local clock.
3. The receivers exchange the observed times at which they received the packet.
4. The clock offset between two receivers is computed as the difference of the local times at which the receivers received the same message.

Fig. 15. Estimation of phase offset in RBS protocol [19].

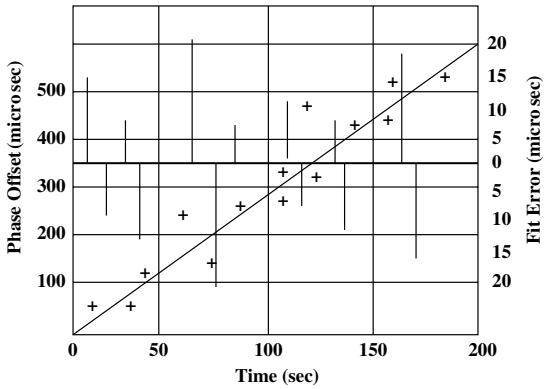

Fig. 16. Clock skew estimation in RBS: each point represents the phase offset between two nodes [19].

scale defined by one node's clock. If $T_{i,k}$ is the time at which receiver i's clock received message k, the coordinates of each point in the figure are defined as follows for each message k received by receivers r_1 and r_2:

$$x = T_{r_1,k}, \quad y = T_{r_2,k} - T_{r_1,k}. \tag{20}$$

The diagonal line drawn through the points represents the best linear fit of the plotted points. The vertical impulses, read with respect to the right-hand Y-axis, show the distance from each point to the best-fit line. Outliers (i.e., the points with the highest residual error) are discarded. A node can compute a least squares error fit (diagonal line) and convert time values between its local clock and that of its peers. Here are some additional features of the RBS protocol [19].

No clock correction. Once clock offset and skew are estimated, the local clocks of nodes are not corrected to run synchronously with a global timescale. Instead, each node keeps a table of parameters that relate the offset and skew of the node with respect to every other clock in the network. Whenever a clock reading is received from another node, a node checks its table to translate the clock value received to the local timescale. The advantage of this method is that considerable energy is saved by avoiding the process of correcting or resetting each node's local clock to a global time.

Post-facto synchronization. The RBS protocol achieves a high level of energy conservation by performing synchronization only when it is needed. Clocks run untethered at their own natural rates and the timestamps from different clocks are compared only when an event of interest occurs. This technique is similar to reactive routing, which we discussed earlier. By synchronizing the nodes only when necessary, energy is conserved because the nodes can be switched to power-saving mode at all other times.

Multi-hop communication. Multi-hop synchronization is required in sensor networks that span several node neighborhoods. Evidently, the possibility of multiple hops would introduce a high degree of variability in message transmission time between multiple receivers of the same message. In this case, the RBS protocol would lose its accuracy. To avoid loss of precision, two nodes located in different neighborhoods are typically synchronized using a third node lying in the intersection of the two neighborhoods. The support for multi-hop communication is more than a convenience; large sensor networks make it a necessity.

Advantages:

1. The largest sources of error (send time and access time) are removed from the critical path by decoupling the sender from the receivers.
2. Clock offset and skew are estimated independently of each other; in addition, clock correction does not interfere with either estimation because local clocks are never modified.
3. Post-facto synchronization prevents energy from being wasted on expensive clock updates.
4. Multi-hop support is provided by using nodes belonging to multiple neighborhoods (i.e., broadcasting domains) as gateways.
5. This protocol is applicable to both wired and wireless networks.
6. Both absolute and relative timescales can be maintained.

Disadvantages:

1. The protocol is not applicable to point-to-point networks; a broadcasting medium is required.
2. For a single-hop network of n nodes, this protocol requires $O(n^2)$ message exchanges, which can be computationally expensive in the case of large neighborhoods.

3. Convergence time, which is the time required to synchronize the network, can be high due to the large number of message exchanges.
4. The reference sender is left unsynchronized in this method. In some sensor networks, if the reference sender needs to be synchronized, it will lead to a significant waste of energy.

4.2. Time synchronization in ad-hoc networks [56]

Romer's synchronization protocol was designed specifically for ad-hoc communication networks [56]. These networks consist of mobile, wireless computing devices that can spontaneously communicate when they are brought within each other's relatively limited transmission range. When this happens, a symmetric bidirectional link is formed between pairs of neighboring nodes and node synchronization takes place, if necessary. Ad-hoc networks are further characterized by the following features. First, nodes are highly dynamic (mobile) and sparsely distributed. Second, the links or connections among the nodes have a relatively short range (e.g., measuring in the order of hundred feet or less) and a short life span, due to node mobility. Third, message passing is possible in the following two ways.

1. Two nodes can exchange messages if one node enters the communication range of the other. This is a direct way of message exchange.
2. Two nodes that reside in different partitions can communicate by using store and forward techniques. An intermediate node can receive the message from the sender, store it temporarily, and eventually forward it to the receiver after entering the receiver's communication range.

Fig. 17 shows a time chart for communication between a sender and a receiver through an intermediate node. At time t_1, node 1 (the sender) sends a message to node 3 (the receiver) through node 2, an intermediate node that stores the message. If node 2 comes into the receiver's transmission range at time t_2, the message is forwarded to node 3.

Romer notes that two fundamental assumptions underlying synchronization in traditional networks no longer hold in ad-hoc networks [56].

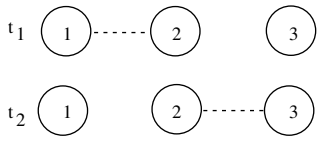

Fig. 17. Store and forward communication [56].

First, in traditional networks, the message transmission delay between any two nodes can be estimated with a high degree of accuracy. Second, in those networks it is possible for nodes to periodically exchange messages in order to synchronize with each other. In ad-hoc networks, the first assumption does not hold because an intermediate node may introduce an arbitrarily long delay in communication between a sender and a receiver. The second assumption does not hold because energy constraints make it impossible to establish a-priori synchronization between arbitrary pairs of nodes in ad-hoc networks.

Romer's protocol is based on two assumptions. The first assumption puts an upper bound, usually denoted by ρ, on the maximum skew of computer clocks. Second, whenever a message is exchanged between two nodes, the connection remains long enough for the two nodes to exchange one additional message.

Similar to RBS, the key idea underlying Romer's protocol is to avoid synchronizing the local clocks of network nodes but instead to generate timestamps using untethered local clocks. When timestamps are passed between nodes, they are transformed to the local time of the receiving node. Since this transformation will generally introduce errors, Romer's protocol seeks to define an upper bound on the absolute value of the clock error received by a node.

When a message containing a timestamp is transferred between nodes, the timestamp is first transformed from the local time to a common time transfer format (UTC) and then to the local time of the receiver. In more detail, the synchronization protocol performs the following steps: (1) determine lower and upper bounds for the real-time elapsed from the generation of the timestamp in the source node to the arrival of the message in the destination node; (2) transform these bounds

to the time of the destination node; and (3) subtract the resulting values from the time of arrival in the destination node. The resulting interval specifies lower and upper bounds for the real-time elapsed from the generation of the timestamp in the source node to the arrival of the message in the destination node. If real-time differences (UTC) are denoted by Δt and computer clock differences by ΔC, the transformation is based on the following equation.

$$(1-\rho) \leqslant \frac{\Delta C}{\Delta t} \leqslant (1+\rho). \tag{21}$$

The equation above can be transformed into:

$$(1-\rho)\Delta t \leqslant \Delta C \leqslant (1+\rho)\Delta t, \tag{22}$$

$$\frac{\Delta C}{1+\rho} \leqslant \Delta t \leqslant \frac{\Delta C}{1-\rho}. \tag{23}$$

It can be inferred that the local clock difference ΔC that corresponds to the real-time difference Δt can be bounded by the following interval:

$$[(1-\rho)\Delta t, (1+\rho)\Delta t]. \tag{24}$$

The basic principles underlying Romer's synchronization protocol are summarized in Fig. 18. In order to transform a time difference ΔC from the local time of node 1 (with an upper bound ρ_1 on its skew) to the local time of node 2 (with upper bound ρ_2), Δt is first estimated by the real-time interval $[\frac{\Delta C}{1+\rho_1}, \frac{\Delta C}{1-\rho_1}]$, which in turn is estimated by the time interval $[\Delta C\frac{1-\rho_2}{1+\rho_1}, \Delta C\frac{1+\rho_2}{1-\rho_1}]$ relative to the local time of node 2. Since an estimate of the lifetime of a timestamp is calculated as the timestamp is passed along different nodes, it is not practical to assume a constant message delay. In general, the transmission delay between any pair of nodes involved in the transmission will be variable. Thus, the message delay between every pair of nodes along the path must be estimated in order to arrive at an accurate solution. The message delay between two nodes is estimated by bounding it within interval $[0, rtt]$ where rtt is the round-trip time between the two nodes.

According to Fig. 19, the delay d for message M_2 can be estimated by the following equation, relative to the sender's clock:

$$0 \leqslant d \leqslant (t_3 - t_2) - (t_6 - t_5)\frac{1-\rho_s}{1+\rho_r}. \tag{25}$$

The difference $t_3 - t_2$ is the round-trip time (rtt), and $t_6 - t_5$ represents the storage time of the message at the receiver. Also, ρ_s and ρ_r are the ρ values for sender and receiver, respectively.

- The goal is to determine estimates of the lifetime of a timestamp.
- Synchronize local clocks of nodes only when an event of interest occurs (similar to reactive routing).
- Generate timestamps to record the time at which an event of interest occured.
- Timestamps are updated by each node using its local clock and the time transformation method.
- The final timestamp is expressed as an interval with a lower bound and an upper bound.

Fig. 18. Principles underlying Romer's synchronization protocol [56].

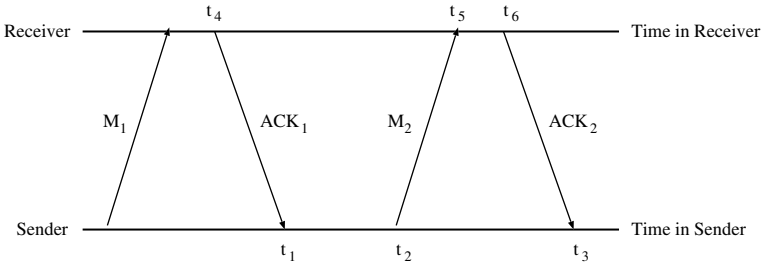

Fig. 19. Message delay estimation [56].

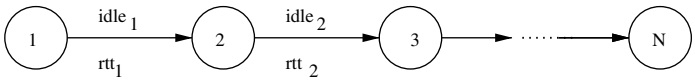

Fig. 20. Timestamp estimation process [56].

As a message is transmitted over a sequence of nodes, the round-trip time *rtt* between each pair of nodes and the idle time (or storage time) are accumulated to estimate the value of the timestamp. A node *n* involved in the transmission must add the times computed by all nodes including the message delay between each pair of nodes in order to estimate the timestamp's value. Fig. 20 further illustrates this mechanism.

Meier et al. recently defined an improved version of Romer's protocol [45]. We briefly discuss this improvement in Section 4.10 below.

Advantages:

1. Local node clocks are allowed to run at their own natural rates, which saves energy by avoiding computer clock correction.
2. The protocol requires low resource and message overhead, making it appropriate for resource-restricted environments.
3. The protocol is suitable for applications that need to communicate over long distances, that is, distances much greater than the nodes' transmission ranges.
4. The time estimation is bounded within an interval.

Disadvantages:

1. The synchronization error increases with the number of hops along the path of the message containing the timestamp. In sparse networks, the number of hops is usually high and this poses an obvious problem unless ρ is quite small.
2. Elson and Romer [20] have claimed that the synchronization achieved by this approach is localized and short-lived. This is appropriate for networks with highly mobile nodes, but it clearly limits the applicability of Romer's protocol.
3. The protocol requires round-trip estimation, which can increase the synchronization error.

4.3. Continuous clock synchronization in wireless real-time applications [49]

Mock et al. [49] defined a protocol for continuous clock synchronization in wireless sensor networks by extending the IEEE 802.11 standard [31] for wireless local area networks. Their protocol improves precision by exploiting the tightness of the communication medium, similar to RBS [19], and also tolerates message loss. The cornerstone of the protocol by Mock et al. is the use of *continuous* clock synchronization. This is in contrast with the IEEE 802.11 standard, which uses *instantaneous* clock synchronization.

In the case of instantaneous synchronization, a node computes a local clock error and adjusts its clock using this computation. This results in abrupt changes in local clock time, which can cause time discontinuity. Time discontinuity can lead to serious faults in distributed systems, such as a node missing important events (e.g., deadlines) or recording the same event multiple times. Ryu and Hong [57] show one such possibility, as seen in Fig. 21. In the figure, assume that a task has a deadline at time 19. Assume further that at

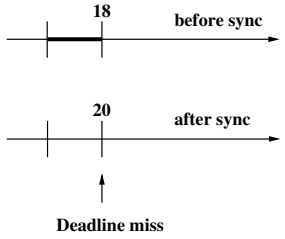

Fig. 21. Instantaneous versus continuous correction [57].

- The master takes its local timestamp at t_1 and broadcasts it at time t_2.
- All slaves physically receive this message at time t_3.
- Each slave adjusts its clock value to match the master's clock at t_4.

Fig. 22. The IEEE 802.11 protocol.

time 18 resynchronization occurs, resulting in local clock being corrected by 2 time units in the forward direction because it lags behind the reference clock by 2 time units. After synchronization, the local clock advances to time 20 and the deadline at time 19 is missed.

Continuous clock synchronization avoids such discrepancies by spreading the correction over a finite interval. The local clock time is corrected by gradually speeding up or slowing down the clock rate. However, this approach suffers from a high run-time overhead since clocks need to be adjusted every clock tick.

The IEEE 802.11 standard [31] includes a master–slave protocol for clock synchronization which achieves limited precision due to instantaneous synchronization. Mock et al. improved the IEEE 802.11 protocol, which we show in Fig. 22, by employing continuous correction and by using an advanced rate adjustment algorithm.

The time-critical path in the IEEE 802.11 protocol is the interval between the master taking its timestamp and the slaves adjusting their clocks

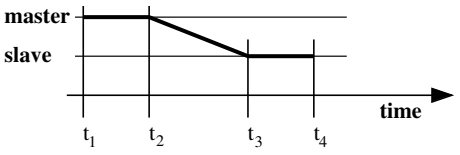

Fig. 23. Time-critical path of IEEE synchronization protocol [49].

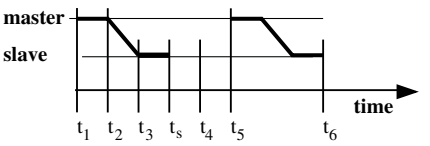

Fig. 24. Reduced time-critical path (from t_3 to t_s) [49].

(i.e., from t_1 to t_4). Fig. 23 illustrates this time-critical path.

Given that the time-critical path above could be quite lengthy, Mock et al. exploit the broadcast property of the medium by assuming that message reception is tight, similar to the RBS protocol [19]. If two receivers receive the same message, it can be assumed that they receive it at approximately the same time. Based on this property, the time critical path is shortened as shown in Fig. 24.

Fig. 25 shows the main steps in the synchronization protocol of Mock et al. The protocol uses two messages for synchronization between a master and a group of slaves, a so-called *indication message* followed by a *confirmation message*. However, subsequent synchronizations are achieved with a single message as the master timestamp for the last confirmation message now serves as the indication message for the next synchronization round. This technique achieves synchronization with just one broadcast message per synchronization round, resulting in a significant reduction in the message overhead.

After the slave nodes complete the estimation of the master's time, they must correct their local

1. The master prepares an *indication* message (time t_1) and broadcasts it (time t_2).
2. The message is delivered to a number of slave nodes with negligible delay, assuming tight message reception.
3. Each slave and the master receive the message (time t_3) and take a local timestamp (time t_4).
4. The master sends its own timestamp in the *confirmation* message (time t_5).
5. Each slave compares the master timestamp with its own timestamp for the reception of the last indication message, computes the difference t_5–t_3, and adjusts its local clock (time t_6).

Fig. 25. Synchronization protocol by Mock et al. [49]. Time instants $t_1,\ldots,t_6$ are relative to Fig. 24.

clocks to reflect the master's clock value. As we discussed earlier, the protocol uses a rate-based algorithm to adjust the virtual clocks of slave nodes to run in tandem with the master. The difference between a virtual and a physical clock can be summarized as follows. The *physical clock* of a node consists of an oscillator, which periodically generates events, and a counter, which records the number of elapsed events. This physical clock is not adjustable in any way. The counter's value cannot be incremented or decremented by the system's software, and its frequency is also fixed. A *virtual clock* is defined as a function from physical clock values to virtual clock values. A virtual clock is intended to correct the skew rate of the physical clock of the slave such that it resembles the physical clock of the master. The function chosen is usually a linear transformation for the sake of simplicity.

Clock correction is performed by correcting the parameters of the linear transformation from physical to virtual clock values with every new synchronization. Thus, changing a virtual clock really means changing the parameters of the transformation from physical to virtual clock values. The goal is to make the virtual clock match the master clock as closely as possible.

Fig. 26 plots the transformation from physical to virtual clock values at a slave node. Function MC relates actual master clock values with a slave's clock values. The goal of the protocol is to estimate this function as accurately as possible. Assuming a constant clock skew between the master's and the slave's clocks, function MC is linear. Thus, this function is shown in Fig. 26 as a straight line. Let SC be the virtual clock of a slave that is corrected at every synchronization point. The values tm_j, tm_{j+k} contained in a message indicate the master timestamps for synchronization messages sm_j and sm_{j+k}, respectively. Values t_j and t_{j+k} indicate the corresponding timestamps on the slave's physical clock. Suppose that t_i is the next synchronization point and that t_i^* is some physical time between t_{i-1} and t_i. T_{i-1} and T_i^* are the corresponding virtual times of the slave clock for physical times t_{i-1} and t_i^*, respectively. Clock correction is performed by adjusting SC at every synchronization point. The adjustment is performed by changing the parameters of the linear transformation function and this adjustment is performed gradually over a period of time (continuous) to avoid time discontinuity. After the point Q in Fig. 26, it can be observed that the master node and the slave are in synchrony, assuming tight reception.

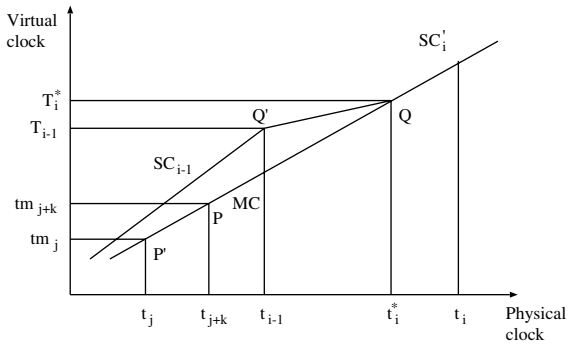

Fig. 26. Time adjustment showing the slave's physical clock on the X-axis and the virtual clock on the Y-axis [49].

The protocol of Mock et al. also provides a high degree of tolerance with respect to message loss. This is an important feature because the rate of message loss is higher in wireless than in wired networks as the wireless medium is rather prone to external interference. The protocol defines an *omission degree* O_D to be an upper bound on the number of consecutive messages that can be lost due to errors in the medium. In order to tolerate up to O_D consecutive message losses, the timestamp values for the last n ($n = O_D + 1$) messages are included in each synchronization message. Thus, a slave can synchronize with the master on the reception of a message, if it has received at least one of the previous n synchronization messages.

Advantages:

1. The protocol provides reasonably good accuracy results when message transmission between master and slaves is tight.
2. The protocol improves over the IEEE 802.11 protocol by using continuous, rather than instantaneous, clock updates [31].
3. The message complexity of the protocol is quite low because the protocol requires only one message per synchronization round.

4. The protocol accounts for potential message losses, a common situation in wireless networks.

Disadvantages:

1. The protocol assumes tight communication between the master and the slaves. Consequently, the protocol cannot accommodate multi-hop communication because of its longer and unpredictable delays.
2. A considerable amount of energy is spent on performing clock correction. This problem is exacerbated by the fact that clock correction is continuous.
3. The computational load on each node is high due to the rate-based correction method.

4.4. Network-wide time synchronization in sensor networks [24]

Scalability is a primary concern in wireless sensor networks because of the large number of nodes with very limited energy resources at each node. The network-wide time synchronization protocol is aimed at ensuring that synchronization accuracy does not degrade significantly as the number of nodes being deployed increases [24]. The objective of the protocol is to establish a unique global timescale by creating a self-configuring hierarchical structure in a wireless network. A node in this structure can simultaneously act as a synchronization server to a number of client nodes and as a synchronization client to another (server) node. The significance of this method lies in achieving synchronization at a network-wide level as opposed to methods which work effectively only within a small cluster of nodes lying in a neighborhood, such as RBS and continuous clock correction [19,49]. The network-wide time synchronization protocol works in two phases: The *level discovery phase*, followed by the *synchronization phase*.

Level discovery phase. The level discovery phase is based on constrained flooding. The root node is assigned level 0; this node initiates this phase by broadcasting a level-discovery packet that contains the identity and the level of the sender. The immediate neighbors that receive this packet assign themselves a level that is one greater than the level in the packet received (i.e., level 1 in this case). After this step, these neighbors broadcast a new level-discovery packet with their own level. This process is continued until each node has a level. Upon being assigned a level, a node neglects further packets to implement a constrained flooding.

Collision handling is important at this point because a node may not receive any *level-discovery* packets due to MAC layer collisions. When a node is deployed, it waits for some time to be assigned a level. If it is not assigned a level within that period, it times out and broadcasts a level-request packet. The neighbors reply to this request by sending their own level and the new node defines its level to be one greater than the level it received.

Synchronization phase. Consider a message exchange between two nodes A and B, as shown in Fig. 27. T_1 and T_4 represent the time measured by A's local clock. Similarly, T_2 and T_3 represent the time measured by B's local clock. We assume that A's level is greater than B's by one. The synchronization phase of the protocol is described in Fig. 28.

Unfortunately, the hierarchical structure that the protocol imposes on the net makes the protocol vulnerable to node failures. This issue must be addressed because nodes can fail unpredictably in a sensor network. When this happens, it is possible for a node at level i not to have a neighbor (i.e., a synchronization server) at level $i-1$. In such cases, the node at level i would not receive an acknowledgement to its synchronization message. To handle this case, the node retransmits a

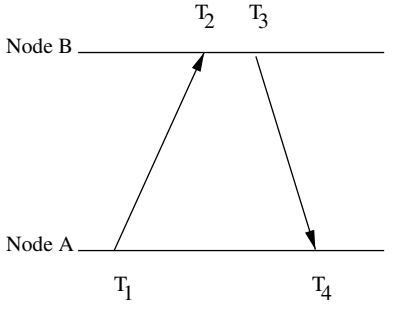

Fig. 27. Message exchange between two nodes A and B [24].

1. At time T_1, A sends a *synchronization-pulse* message to B. The synchronization-pulse message contains the level number of A and the value of T_1. Node B receives this packet at T_2, where $T_2 = T_1 + \delta + d$ and δ represents the clock offset between the two nodes and d represents the propagation delay.
2. At time T_3, B sends an $acknowledgement$ packet to A. This packet contains the level number of B and the values of T_1, T_2, and T_3. With this information, node A calculates the clock offset and delay as follows.

$$\delta = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}; d = \frac{(T_2 - T_1) + (T_4 - T_3)}{2} \tag{26}$$

3. Node A corrects its clock to synchronize with node B, based on the computed offset.

Fig. 28. Synchronization protocol for network-wide time synchronization protocol [24].

message for a fixed number of times before assuming that it has lost all its neighbors. If the node does not receive any responses to its synchronization messages, it broadcasts a *level-request* packet with a new level. Next, the node synchronizes with its new neighbors as described in Fig. 28. The number of retransmissions of a synchronization message is subject to two constraints. If this number is too large, it will increase the time taken for synchronization (i.e., the convergence time). If the number is too small, a node may erroneously conclude that its server has died. In this case, the protocol will cause unnecessary message flooding in the network. The authors suggest that an optimal number of retransmissions is four [24].

Advantages:
1. The protocol is scalable and the synchronization accuracy does not degrade significantly as the size of the network is increased.
2. Network-wide synchronization is effectively achieved in contrast with the protocol by Mock et al. [49], which works effectively only within a small cluster of nodes.
3. Network-wide synchronization is computationally less expensive when compared to such protocols as NTP [48].

Disadvantages:
1. Energy conservation is not very effective because it requires a physical clock correction to be performed on local clocks of sensors while achieving synchronization.
2. The protocol requires a hierarchical infrastructure which makes it unsuitable for applications with highly mobile nodes.
3. Support for multi-hop communication is not provided.

4.5. Delay measurement time synchronization protocol [54]

Time-keeping is the process of maintaining a uniform notion of time among the sensors that participate in the network [54]. A global timestamp provides a basis for merging individual sensor readings into a database. Also, synchronized time is essential for energy-efficient scheduling and power management. This protocol, which appears in Fig. 29, includes the following features.

1. Maintenance of a local clock.
2. Synchronization of local clocks over the whole network to create a network time.

1. A leader is chosen among a set of communicating nodes.
2. The leader broadcasts its local clock value to the other nodes.
3. All receiving nodes compare their local clock values relative to the leader's time.
4. If the delays in the path from one node to another node can be estimated accurately, the two nodes can be synchronized.
5. When the message is broadcast, the sender of the packet will be synchronized with all the nodes that receive its packet.

Fig. 29. Algorithm underlying flexible lightweight time-keeping protocol [54].

3. Synchronizing to a global time source by connecting the global source to the synchronization leader in the network.
4. Application programming interface for providing services to client applications.

This protocol has been implemented on sensor nodes consisting of Berkeley motes running the TinyOS kernel [61]. Node synchronization is based on the concepts of *event timestamps* and *network event scheduling*. The synchronization protocol specifically combines delay measurement with the property of the sender's timestamp being a common-view timestamp from the receivers' point of view. The receivers can synchronize with each other better than they can synchronize with the sender.

The synchronization accuracy of this protocol is bounded mainly by the precision of delay measurements along the path. Since only one message is required to synchronize all nodes within the leader's transmission range, this method is quite energy efficient. It is also computationally lightweight because there are no complex mathematical operations involved.

Multi-hop synchronization is also supported by extending the single-hop synchronization protocol as follows. If a node knows that it has children, it sends a broadcast time signal after adjusting its own time. The node can now synchronize with its children by using single-hop time communication with a known leader. To tackle the problem that in most networks nodes have no knowledge about their children, the concept of a time-source level is used to identify the network distance of a node from the master. A time master initiates the synchronization protocol; the master's time source level is zero. A node synchronizing directly with the master is at time source level one. This algorithm guarantees that the root time will be propagated to all network nodes with a limited number of broadcasts.

Advantages:
1. A user application interface is provided to monitor a wireless sensor network at run-time.
2. Computational complexity is low and energy efficiency is quite high.

Disadvantages:
1. The protocol can be applied only to low resolution, low frequency external clocks.
2. Synchronization accuracy is traded for low computational complexity and energy efficiency.

4.6. Probabilistic clock synchronization service in sensor networks [53]

Most synchronization protocols in practice rely exclusively on deterministic algorithms. An advantage of deterministic methods is that they usually guarantee an upper bound on the error in clock offset estimation. However, when the system resources are severely constrained, a guarantee on synchronization accuracy may result in a large number of messages being exchanged during synchronization. In these cases, probabilistic algorithms can provide reasonable synchronization accuracy with lower computational and network overhead than deterministic protocols. Pal-Chaudhuri et al. [53] defined an extension to RBS [19], by providing a probabilistic bound on the accuracy of clock synchronization. An attractive feature of their protocol extension is that it allows to tradeoff dynamically synchronization accuracy for computational and energy resources.

As we show in Fig. 16, in RBS [19] multiple messages are sent from the sender to the set of receivers, and the differences in actual reception times at the receivers are plotted. As these messages are independently distributed, the difference in reception times gives a Gaussian (or normal) distribution with zero mean. Assuming a Gaussian probability distribution for the synchronization error, the relationship between a given maximum error in synchronization and the probability of actually synchronizing with an error less than the maximum error can be easily computed.

If the maximum error allowed between two synchronizing nodes is $\epsilon_{\max}$, then the probability of synchronizing with an error $\epsilon \leqslant \epsilon_{\max}$ is derived from the Gaussian distribution property,

$$P(|\epsilon| \leqslant \epsilon_{\max}) = \frac{\int_{-\epsilon_{\max}}^{\epsilon_{\max}} \epsilon^{-x^2/2}\,dx}{\sqrt{2\pi}}. \quad (27)$$

As the ϵ_{max} limit is increased, the probability of failure $(1 - P(|\epsilon| \leqslant \epsilon_{max}))$ decreases exponentially.

Based on this relation between the maximum synchronization error and the probability of actually synchronizing with a smaller error than the predefined maximum, PalChaudhuri et al. [53] derive expressions that convert service specifications (maximum clock synchronization error) to actual protocol parameters (number of messages and synchronization overhead). The probability of the achieved error being less than the maximum specified error is given as follows:

$$P(|\epsilon| \leqslant \epsilon_{max}) = 2\mathrm{erf}\frac{\sqrt{n}\epsilon_{max}}{\sigma}. \quad (28)$$

In the above equation, n is the minimum number of synchronization messages to guarantee the error and σ is the variation of the distribution.

The relationship between the number of messages, n, and error probability P is shown in Table 4. The table reports data for ϵ_{max}/σ ratios of 0.5, 1.0, and 2.0.

Advantages:

1. A probabilistic guarantee reduces both the number of messages exchanged among nodes and the computational load on each node.
2. A tradeoff between synchronization accuracy and resource cost is allowed.
3. Support for multi-hop networks, which span several domains, is provided.

Table 4
Variation in probability and number of messages for different values of ϵ_{max}/σ [53]

ϵ_{max}/σ	Probability	Number of messages
0.5	0.95	16
0.5	0.99	28
0.5	0.999	44
1.0	0.95	4
1.0	0.99	7
1.0	0.999	11
2.0	0.95	1
2.0	0.99	2
2.0	0.999	3

Disadvantages:

1. In safety-critical applications (e.g., nuclear plant monitoring), a probabilistic guarantee on accuracy may not suffice.
2. The protocol is sensitive to message loss; however, provisions for message loss are not considered.

4.7. Sichitiu and Veerarittiphan's protocol [58]

Sichitiu and Veerarittiphan's protocol [58] provides deterministic clock synchronization for wireless sensor networks with minimal computational and storage complexity. The protocol is especially suitable for applications with severe constraints on computational power and bandwidth. It uses two algorithms called *mini-sync* and *tiny-sync*. The *tiny-sync* algorithm acquires its name from the fact that it needs very limited resources, fewer resources than *mini-sync*.

The two algorithms have various common features.

1. A tight deterministic bound is provided for clock offset and skew, as opposed to probabilistic guarantees [53].
2. The protocols are highly tolerant to message losses.
3. Both protocols have low computational and storage complexity.
4. Both protocols can be extended to any communication network that allows bidirectional data transmission.
5. The estimation of clock skew and offset is performed using the set-valued estimation method [40] discussed in Section 2.4.4.

Recalling Eq. (17) of the set-valued estimation method, for nodes 1 and 2, the skew and offset between their clocks are captured by the following equation, where a_{12} and b_{12} represent the skew and offset between the clocks in node 1 and node 2.

$$t_1(t) = a_{12}t_2(t) + b_{12}. \quad (29)$$

Sichitiu and Veerarittiphan's protocol [58] extends the set-valued estimation method [40] in two significant ways.

1. *Relaxing the immediate reply assumption*. In Fig. 11, it is assumed that node P_j immediately responds to node P_i. The correctness of the approach is not affected if P_j sends a delayed reply. However, if the delay between $\underline{t_{ik}}$ and $\overline{t_{ik}}$ increases, the precision of the estimates will decrease. Since P_j can delay its reply in practice, the loss of precision is handled by letting P_j timestamp the message both upon receipt (t_{jkr}) and when it resends it (t_{jkt}). This gives us two data points ($\underline{t_{ik}}$, t_{jkr}, $\overline{t_{ik}}$ and $\underline{t_{ik}}$, t_{jkt}, $\overline{t_{ik}}$) which will be treated independently. The obvious solution is to choose the data point that gives a better precision.
2. *Increasing accuracy by considering the minimum delay*. If the minimum delay that a message encounters between two nodes is known, the data points can be adjusted for a boost in precision. In Fig. 11, if the delay between the time P_i timestamps the message (t_{i1}) and the time at which P_j timestamps the message (t_{j1}), as well as the delay between P_j timestamping the message and P_i receiving that timestamp are known, we can use this information in the data triplet to make more estimates.

Tiny-sync: The *tiny-sync* algorithm is based on the observation that not all data points obtained from the set-valued estimation method are useful. Each data point consists of two constraints, which are bounds on the clock offset and clock skew. At any point of time, only four constraints are kept instead of six constraints. Upon the arrival of a new data point, the two new constraints are compared with the existing data points and only the four constraints that result in the best estimates are kept.

The computational complexity of the *tiny-sync* algorithm is relatively low because the comparisons to determine the four best constraints involve only a few arithmetic operations. In addition, the storage space required is also quite low because at any time, only four constraints and eight timestamps (two per constraint) need to be stored.

Mini-sync: The *mini-sync* algorithm improves the accuracy of *Tiny-sync* at a small computational cost. In Fig. 12, when the data point that corresponds to $[\underline{t_{i3}}, \overline{t_{i3}}]$ arrives, we can ignore the data point that corresponds to t_{i2} and consider only the first and third data points. However, when the fourth data point arrives, since we discarded the second data point, we are forced to use the first and the fourth data points as bounding constraints. Although the second data point—when bound to the fourth data point—could have yielded better precision, this precision cannot be accomplished because the second data point was discarded. *Mini-sync* corrects this flaw and improves precision by discarding a constraint only if a newer constraint eliminates an existing constraint. This means that the constraints corresponding to the second data point will not be discarded when the third data point arrives. It will instead be saved until the fourth data point uses it to obtain a better estimate. In practice, experiments reveal that around 40 data points (80 constraints) have to be stored at a time, which is quite reasonable.

Advantages:
1. The protocols provide a tight, deterministic synchronization scheme with low storage and computational complexity.
2. The protocols are suitable for sensor networks that are highly constrained in bandwidth and computational power.
3. The protocols are tolerant of message losses.

Disadvantages:
1. The scalability and robustness of the protocols have not been discussed.
2. The convergence time, which is the time needed to achieve synchronization of the entire network, is high.
3. The sensor network is logically organized as a hierarchy, making it inapplicable to mobile sensor networks.

4.8. Time-diffusion synchronization protocol [62]

The Time-Diffusion synchronization Protocol (TDP) enables all the sensors in the network to have a local time that is within a small bounded time deviation from the network-wide "equilibrium" time. Due to clock skews, the algorithms within the protocol have to be applied periodically.

Hence, the protocol operates in alternating *active* and *inactive phases*. The protocol is comprised of several algorithms, which will be described next in the context of one such *active* phase.

Within each *active phase* there are multiple *cycles*, each cycle lasting a duration τ. In each cycle, a subset of the nodes are elected as the masters by an Election/Relection Procedure (ERP). Each master concurrently initiates a diffusion of timing messages; these messages effectively create a tree-like propagation structure dynamically in the network, for each diffusion. The non-leaf nodes in this tree are the nodes which propagate the timing messages, and are termed as "diffused leaders". These diffused-leader nodes are also elected by the ERP. Thus, it may happen that a node does not qualify to be a diffused leader node, and hence will not propagate the diffusion. The goals of the ERP are twofold.

- To eliminate outlier nodes whose clock variance is above some threshold function based on a specific type of variance calculation, termed the *Allen variance*. This variance is determined by exchanging messages and calculating deviations between pairs of adjacent nodes using a *Peer Evaluation Procedure* (*PEP*).
- To achieve load distribution among the nodes because the roles of masters and diffused leaders put a greater demand on the energy resource. The load distribution is achieved by taking turns at being the master, based on factors such as the available energy level being above a tunable threshold.

In each cycle, the diffusion of timing messages helps to converge the local times, and reach a common notion of the system-wide time.

Each cycle has *two logical functions*, executed serially—(1) determining master nodes and diffused leader nodes, using the PEP, and then (2) the main *Time Diffusion Procedure (TP)*. Each *cycle* has duration τ; the TP consists of multiple *rounds*, initiated δ time units apart. The timing relations are illustrated in Fig. 30. There is a single broadcast within each round, with respect to a single master. Note that each master initiates a concurrent broadcast that gets diffused in that round

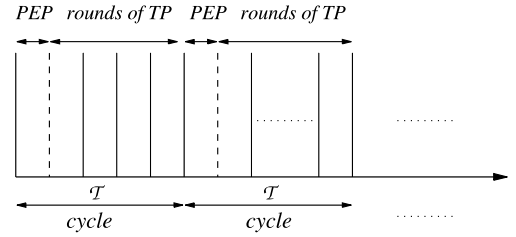

Fig. 30. Illustration of timing relationships between the TP rounds (each of duration δ) and the PEP duration within each cycle, and the various cycles, each of duration τ, within an active phase.

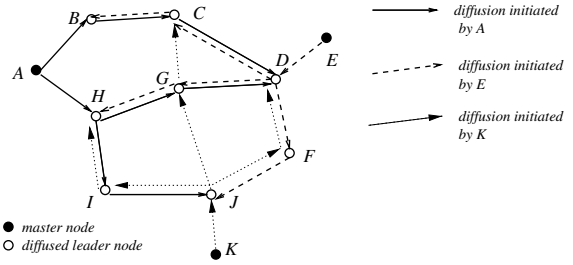

Fig. 31. Illustration of time diffusion with three master nodes and $n = 3$ hops. The level of a node is defined with respect to each master. Outlier nodes, which do not diffuse timing messages, are not shown for simplicity. In each round, nodes take the hop-weighted (or cumulative deviation weighted) average of the different times received from the masters' diffused broadcasts.

in a tree-like structure, as illustrated in Fig. 31. Also note that the masters' time can be coordinated to an external precise time server that does periodic broadcasts of a reference time. If no time servers are available, the protocol works equally well by using a time that is independent of the time used by the Internet, e.g., Universal Coordinated Time (UTC). We now look at the details of a single cycle with respect to a single master.

1. The first function (PEP) in each cycle is to determine the master node eligibility for the next cycle, and the diffused leader responsibility for the remainder of this cycle.
   (a) The first step occurs between any master, which is at level one, and its neighbors.
      (i) The master sends a large number of timestamped scan messages to its neighbors.

(ii) The neighbors send back acknowledgements containing the 2-sample Allen variance of the local clock from the master's clock.
(iii) Based on the received samples, the master calculates (a) an outlier ratio γ_{yz} for itself (y) and each neighbor z, (b) the average of the Allen variances, and (c) the average of the Allen deviations. Now, (a), (b), and (c) are sent to each neighbor z in a RESULT message.

(b) In each subsequent step $j = 2, 3, \ldots, n$, the above is repeated between each level j diffused leader node and its neighbors.

As this broadcast diffusion progresses, all sensors will get the outlier ratios and the average Allen deviation (with respect to their neighbors). These are used to evaluate the quality of their clocks with respect to their neighbors. If a node's average outlier ratio is >1, its local clock deviates from the clocks of its neighbors by more than twice the Allen variance. In this case, that node does not become a diffused leader during the (Time Diffusion Procedure of the) current cycle, or a master in the next cycle.

Further, among the nodes that are eligible for being masters in the next cycle, whether a node will actually qualify for being a master in the next cycle now depends on its energy availability being above a certain (dynamically adjustable) threshold. Load balancing is done by rotating the role of master nodes; hence the algorithm can be viewed as being distributed. Analogously but somewhat differently, whether the nodes eligible to become diffused leaders in the current cycle actually assume that role is determined dynamically for each round in this cycle, based on energy level considerations.

2. The TP performs the main function of diffusing the time from each master in a tree-like manner for n hops, where n is some predetermined parameter smaller than the diameter of the network. It uses the message $M(t_{M,i}, n, \beta_{M,k})$, where
   - $t_{M,i}$ is the diffused time of the master M, to which the nodes synchronize in round i.
   - n is the number of levels (i.e., depth) to which the timing information is to be diffused.
   - $\beta_{M,k}$ is the deviation of the corresponding $t_{M,i}$ at a node k hops from the master M.

The TP within any cycle has a succession of rounds initiated by the master, δ time units apart from $t_{M,0}$. The broadcast within a round completes before the next round is initiated. The following broadcast is executed for each round i.

(a) The first step, executed between any master, which is at level one, and its neighbors:
   (i) Send a timing message $M(t_{M,i}, n, \beta_{M,k})$ to neighboring nodes at time $t_{M,i}$.
   (ii) Elected diffused leaders at the next level respond with a timestamped ACK message.
   (iii) Master computes $\Delta = average(\Delta_j)$, where Δ_j is the round-trip time between the master and the diffused leader j. The diffused time from the master node is

$$t_{M,i} = t_{M,i} + \Delta/2 + \delta. \tag{30}$$

   Here, δ is the amount of time that the nodes wait (relative to $t_{M,i}$) before adjusting their clocks at the end of the round. The standard deviation α of the *rtt* Δ_j, which gives an estimate of the quality of diffused time $t_{M,i}$, is accumulated in $\beta_{M,k}$ at each hop from the master. This accumulated deviation is

$$\beta_{M,k} = \beta_{M,k-1} + \alpha, \tag{31}$$

   where $k \leqslant n$ is the number of hops from the master.

(b) On receiving a timing message $M(t_{M,i}, n, \beta_{M,k})$, for each subsequent step $j = 2, 3, \ldots, n$, the above is repeated between the elected diffused leaders at level j, and their neighbors.

For each round, each node builds a table *Table* with rows of the following format, and populates it with the information received within that round.

$$\langle master\_id\ M, \beta_{M,k}, t_{M,i} \rangle. \tag{32}$$

After each round i within a cycle, each node resets its time to t_i, the weighted sum of the times

$t_{M,i}$ in its table, based on the values of all the diffused messages received in this round from different master initiators, and as recorded in the local *Table*. The table is also cleared at the end of each round i.

$$t_i = \sum_{\langle \beta_{M,k}, t_{M,i} \rangle \text{ in } Table} \frac{[\sum_{\beta_{M',k'} \text{ in } Table} \beta_{M',k'}] - \beta_{M,k}}{\sum_{\langle \beta_{M'',k''}, t_{M'',i} \rangle \text{ in } Table} [[\sum_{\beta_{M',k'} \text{ in } Table} \beta_{M',k'}] - \beta_{M'',k''}]} * t_{M,i}. \quad (33)$$

For any $t_{M,i}$, (i) the numerator of the weight is the sum of all the deviations of all the diffusion messages, less the deviation for this particular message, and (ii) the denominator of the weight is the sum of all such numerators for all the timing entries in the table. Thus, the value of each clock is set to the weighted average of the clock values of the different master nodes of that round, after averaging the data collected from multiple messages received within that round. Due to the weighted averaging, all the nodes tend towards a common equilibrium time.

Advantages:

1. The protocol is tolerant of message losses.
2. The protocol achieves a system-wide ''equilibrium'' time across all nodes, computed using an iterative weighted averaging technique, and involves all the nodes in the synchronization process.
3. The diffusion does not rely on a static level-by-level transmission. This non-dependence on a static structure provides flexibility and fault-tolerance.
4. The protocol is geared towards mobility.
5. Although there is a hierarchical structure, that is neutralized by having multiple master nodes distributed across the network.
6. Most synchronization protocols require precise time servers and cannot function properly without these servers. On the other hand, the TDP protocol can provide synchronization even without external time servers.

Disadvantages:

1. Each active period has multiple cycles, and each cycle has multiple rounds, in each of which a diffusion broadcast is initiated by multiple masters. This leads to high complexity.
2. The convergence time tends to be high when no external precise time servers are used. However, if the servers are used, the convergence time is comparable to a server-based technique.
3. It appears that it is possible for clocks to run backward. This can happen whenever a clock value is suddenly adjusted to a lower value.

4.9. Asynchronous diffusion protocol [42]

Li and Rus have defined a so-called *rate-based diffusion* protocol in which nodes achieve synchronization by flooding their neighbors with information about each node's local clock value [42]. After each node has learned the clock values of all its neighbors, the node can use a mutually agreed upon *consensus* value to adjust its clock. Examples of consensus values suggested by the authors include the highest clock reading in the net, the lowest clock reading, or some statistical value based on the clock readings (e.g., the average or the median of the readings). According to the authors, using the highest or the lowest reading yields the simplest synchronization algorithm; however, this strategy lacks robustness. A malicious or erratic node may impose an abnormally high (or low) clock value on the whole network.

Li and Rus define a *synchronous* and an *asynchronous* version of their rate-based diffusion protocol [42]. Fig. 32 illustrates the synchronous version. The algorithm appearing in the figure is assumed to be executed with a certain frequency by all nodes contained in the network. During each synchronization round, each network node n_i exchanges its clock reading with every neighbor n_j. Node n_i adjusts its clock value by a factor proportional to $t_i - t_j$, the difference in the clock values of n_i and n_j. Coefficient r_{ij} is the so-called *diffusion value* of node n_j relative to n_i; this value indicates the weight of n_j when adjusting n_i's clock value.

1. **for each** node n_i in network $\mathcal{N}$ **do**
2. Exchange clock reading between n_i and its neighbors in $\mathcal{N}$
3. **for each** neighbor n_j **do**
4. Let t_i and t_j be the readings of n_i and n_j
5. Adjust n_i's clock to be $t_i - r_{ij} \cdot (t_i - t_j)$.

Fig. 32. Algorithm underlying the synchronous version of Li and Rus's diffusion protocol [42].

Li and Rus define the r coefficients in such a way that $\forall i \forall j$, $r_{ij} \geqslant 0$ and also $\sum_{\forall j \in \mathcal{N}} r_{ij} = 1$. Also, the coefficients are symmetric, meaning that $\forall i \forall j$, $r_{ij} = r_{ji}$. When nodes n_i and n_j are not in the same neighborhood, they cannot exchange clock readings. In this case, the coefficients are assigned values $r_{ij} = r_{ji} = 0$. Li and Rus showed that this protocol converges to the average value of the clock readings in the network, within a certain error, in a bounded number of synchronization rounds [42]. The number of synchronization rounds depends on the actual values of the coefficients in the network.

In the *asynchronous* version of Li and Rus's diffusion-based protocol, nodes compute average clock readings asynchronously with respect to other network nodes. (In the synchronous version, nodes exchange clock values and adjust their clocks simultaneously.) Fig. 33 illustrates the algorithm underlying the asynchronous version of the diffusion protocol.

Any network node can now update its clock value asynchronously with respect to other nodes in the network. As we show in Fig. 33, a node n_i starts a round of synchronization by first asking all its neighbors for their clock values. Next, the node computes the average of its clock value and the values obtained from the neighbors. Finally, the node updates its clock to the computed average and notifies its neighbors of the computed value. Li and Rus show that the asynchronous version of the protocol converges to the average value of clock readings in the network, under relatively broad assumptions on network connectivity, provided that all net nodes perform synchronization with a certain probability. Nodes are not required to be fully connected with all other nodes in the network in order for this protocol to converge, although the network must be connected at all times [42]. Evidently, if a node or node subset becomes permanently disconnected from the rest of the network, network-wide clock synchronization is impossible.

Li and Rus ran many simulations of the asynchronous synchronization algorithm while varying several synchronization parameters. Some simulations show that the synchronization error among network nodes decreases exponentially with the number of synchronization rounds for a network of 200 nodes [42]. In addition, Li and Rus analyzed the convergence speed of the protocol as a function of node density (i.e., the number of nodes in a neighborhood). These simulations show that sparse networks (i.e., networks with few nodes in a given area) exhibit both a slower convergence speed and a high variation in convergence time. Networks with a high node density converge faster, and with lower variation, than sparse networks. Similarly, the convergence speed improves as the communication range of the nodes is increased.

Advantages:

1. The protocol achieves a system-wide "equilibrium" time across all nodes involved in a synchronization.
2. The protocol does not rely on an external time server or a synchronization leader to reach convergence, which is beneficial for the robustness of the protocol.

Disadvantages:

1. The protocol seems to violate a fundamental requirement of clock synchronization, namely, that time never run backward. (See Fig. 3 above.) As nodes adjust their clock values

1. **for each** node n_i in network $\mathcal{N}$ with uniform probability **do**
2. Ask n_i's neighbors for their clock values
3. Compute average value of all readings, including n_i's
4. Adjust n_i's value to the computed average and send the new value to all n_i's neighbors

Fig. 33. Algorithm underlying the asynchronous version of Li and Rus's diffusion protocol [42].

down, they will repeat clock values obtained earlier, defeating one purpose of synchronization. This problem is exacerbated in the asynchronous version of the protocol because in this case, nodes can adjust their clock values independent of other nodes in the network.
2. The protocol assumes that any two nodes can accurately swap their clock readings when running their synchronization protocol. In practice, achieving this goal can be quite difficult. Li and Rus advocate the use of the RBS protocol [19] for this purpose, because RBS can in fact achieve a high level of accuracy.
3. The complexity of the protocol is quite high, comparable to that of RBS. Many synchronization rounds are needed to reach reasonable convergence, and each node must communicate with every other node in its neighborhood to achieve convergence. Worse yet, in Li and Rus's simulations, hundreds of synchronization rounds are needed to achieve reasonable accuracy [42]. It is unclear whether the asynchronous diffusion protocol can improve RBS's synchronization accuracy.
4. The protocol makes no provisions for exploiting clock skews in achieving synchronization. Given that all node pairs exchange clock information during each synchronization round, it would be relatively easy for nodes to estimate the skew of their neighbors' clocks in an effort to improve accuracy.

4.10. Other protocols

Clock synchronization in sensor networks is currently the subject of numerous active investigations. For reasons of space, we are unable to discuss in detail all the new protocols that have been defined recently. In this subsection, we briefly summarize some of those protocols.

4.10.1. Tulone's Clock Reading protocols

Tulone sketched a deterministic protocol and a probabilistic protocol for clock synchronization in sensor networks [67]. The first protocol, called *Deterministic Clock Reading* (DCR), seeks to minimize the offset of a node's clock relative to other nodes in a sensor network when the node is out of the communication range of any other network node. Assume that synchronization rounds, during which nodes correct their clock values, occur periodically in a sensor network. If a node becomes temporarily isolated from the network, it will miss one or more synchronization rounds with neighboring nodes. In this case, the offset of the node's clock relative to the clock of its neighbors may grow beyond an acceptable threshold.

The DCR protocol observes the standard deviation of a node's clock with respect to one of the node's neighbors over two consecutive synchronization rounds. The observed deviations can be used to correct the speed of a node's clock, similar to Cristian and Fetzer's *calibrated-clocks* method [12]. In the world of sensor networks, the rationale of the DCR method is similar to that of the set-valued estimation protocol, which we discussed in Section 2.4.4, in that DCR adjusts a node's clock rate based on multiple readings of a neighbor's clock. As with set-valued estimation, it is unclear whether DCR will work in practice, because it is possible for a node to overcorrect its clock rate. This may happen when a node overestimates the speed of a neighbor's clock, for instance, because of unpredictable variations in message transmission delays. When this happens, a chain of corrections involving all nodes in an entire neighborhood may ensue, causing all nodes to increase indefinitely their clock rates.

The second protocol, *Probabilistic Clock Reading* (PCR) is a probabilistic variation of DCR. PCR overcomes the problem of intermittent communication by using the theory of time-series forecasting. A network node uses a time-series approximation of a sequence of clock readings from one of the node's neighbors in order to estimate the offset and skew of the neighbor's clock relative to the node's clock [67].

4.10.2. Meier et al.'s protocol

Meier et al. recently defined an improved version of Romer's protocol [45] which we discussed in Section 4.2. Similar to Romer's protocol, Meier et al. seek to provide tight lower and upper bounds on the clock reading of local node n_i when an event

is detected by a different node n_j. Meier et al. define a tighter lower bound for n_i's clock reading using a different formula from Romer's protocol. In addition, Meier et al. define a method for defining optimal bounds under the assumption that message delay uncertainties are negligible [45]. Under this assumption, multiple consecutive message exchanges between two nodes can be regarded as a single communication event. Although these assumption are not true in practice, the method of Meier et al. can be modified to accommodate small communication delays, such as those achieved with RBS [19].

4.10.3. Lightweight Tree-based Synchronization

Van Greunen and Rabaey's *Lightweight Tree-based Synchronization* (LTS) protocol [68] is a slight variation of the network-wide synchronization protocol of Ganeriwal et al. [25] which we discussed in Section 4.4. Similar to network-wide synchronization, the main goal of the LTS protocol is to achieve reasonable accuracy while using modest computational resources (both in terms of memory space and CPU time). Van Greunen and Rabaey give two versions of the LTS protocol [68]. In the *centralized* version, each round of synchronization is initiated by a designated node at some frequency. In the *decentralized* version, any node can start a synchronization round. As with Network-Wide Synchronization [25], the LTS protocol seeks to build a tree structure within the network. Adjacent tree nodes exchange synchronization information with each other. A disadvantage is that the accuracy of synchronization decreases linearly in the depth of the synchronization tree (i.e., the longest path from the node that initiates synchronization to a leaf node). Van Greunen and Rabaey discuss various ideas for limiting the depth of the tree; the performance of both protocol versions is analyzed with simulations [68].

4.10.4. TSync protocol

Similar to Van Greunen and Rabaey's LTS protocol, Dai and Han's *TSync* protocol [15] is based on the network-wide synchronization protocol of Ganeriwal et al. [25]. As with LTS, *TSync* has a centralized version, called the *Hierarchical Referencing Time Synchronization* (HRTS) protocol, and a decentralized version, called the *Individual Time Request* (ITR) protocol. The HRTS protocol cleverly combines the notion of hierarchical synchronization, typical of network-wide synchronization, with receiver-to-receiver synchronization, similar to RBS [19]. Dai and Han further enhance the performance of both the HTRS and ITR protocols by using dedicated MAC-layer channels for synchronization. The ITR protocol differs from the HRTS protocol in that synchronization is initiated by any node as opposed to a designated base station. Dai and Han compared empirically the performance of the HRTS and ITR protocols with a multi-hop extension of RBS [19]. In Dai and Han's experiments, HRTS can achieve a synchronization accuracy close to that of the RBS extension, while reducing the total number of exchanged messages with respect to RBS. However, the accuracy obtained with both RBS and HRTS, in the order of 20 μs for single-hop synchronization, is lower than other reported results for RBS. (See, e.g., [19].) The performance of the ITR protocol is worse than both HRTS and RBS, especially in the case of multi-hop synchronization.

4.10.5. Hu and Servetto's protocol

Finally, Hu and Servetto [30] defined a protocol for synchronization in networks with a high concentration of nodes per unit of surface. Synchronization proceeds in concentric waves, starting from a master node located in the center of the network. Under strong assumptions on the behavior of the net, Hu and Servetto show that their protocol is optimal in the sense that all nodes will eventually synchronize with the master's clock. This is a valuable theoretical result; however, it is currently unclear how this protocol will perform in practice. Hu and Servetto assume that there is no communication delay between nodes (i.e., message transmission delay is always zero) and no conflicts on network use (i.e., a node will have immediate access to the network whenever it needs to transmit a message). In addition, the complexity of the protocol seems to be high, meaning that achieving optimality involves a large number of message exchanges among network nodes.

5. Comparison of protocols

We compare and evaluate the various synchronization protocols. Before we evaluate the various protocols, we must first define in detail the criteria that we will use in our comparisons. For the sake of clarity, we divide our evaluation criteria between quantitative and qualitative criteria. The former include synchronization accuracy, computational complexity, and convergence time. The latter include scalability, energy efficiency, and fault-tolerance. When taken together, these measures provide a good characterization of the applicability and performance of each protocol.

5.1. Quantitative evaluation

We note at the outset that the protocols differ broadly in their computational requirements, energy consumption, precision of synchronization results, and communication requirements. In addition, no protocol clearly outperforms the others in all possible applications of wireless networks. Rather, it is quite likely that the choice of a protocol will be driven by the characteristics and requirements of each application. For instance, a low-cost, low-precision protocol could be appropriate for many environmental monitoring applications. However, many safety-critical applications, such as aircraft navigation or intrusion detection in military systems, will demand high-precision protocols in order for nodes to correctly identify events occurring in the net and for an application to respond to those events.

Synchronization precision. Each network node has a physical clock consisting of hardware oscillator circuits. Unfortunately, the frequency of hardware clocks varies from one node to another within a specified range. Thus, clocks on different nodes in wireless networks operate at different rates. Consequently, the clock values used for synchronization in wireless networks are not physical clock readings. Instead, network nodes generally use a logical notion of clocks and time. Logical clocks can be modified both by software (e.g., during synchronization) and hardware (e.g., by the physical clocks). Consequently, synchronization precision can be defined in two ways.

1. *Absolute precision*. The maximum error (i.e., skew and offset) of a node's logical clock with respect to an external standard such as UTC.
2. *Relative precision*. The maximum deviation (i.e., skew and offset) among logical clock readings of the nodes belonging to a wireless network.

In our discussions, we generally use the notion of relative precision (2) above, unless otherwise noted. In general, high synchronization precision is clearly a desirable feature of a synchronization protocol. However, in the protocols that we studied, higher synchronization precision comes at the expense of increased computational cost measured in terms of algorithmic complexity, the number of messages exchanged among nodes, and the storage requirements of the protocol. The quantitative precision of the various protocols appears in Table 5.

Piggybacking. Piggybacking is the process of combining the acknowledgement messages during synchronization with messages that carry synchronization data among nodes. Instead of sending independent acknowledgement messages, these messages are piggybacked on the data messages that have to be sent to the node, in order to reduce message traffic in the network. Piggybacking is clearly advantageous because wireless networks are often subject to severe bandwidth constraints and piggybacking alleviates communication demands on the network. In addition, piggybacking can also reduce the storage requirements on network nodes because storage space is also saved by clubbing acknowledgements with data messages. Romer's protocol [56] and network-wide time synchronization [24,25] use piggybacking.

Computational complexity. As wireless networks often have limited hardware capabilities and severe energy constraints, the complexity of a synchronization protocol can make a protocol impractical for many applications. Here we distinguish between the computational complexity of a protocol (i.e., its run-time and memory requirements) from the message complexity (i.e., the number of messages exchanged during synchronization). (See also discussion on convergence time below.)

Table 5
Quantitative performance comparison of synchronization protocols

Protocols	Factors for performance comparison						
	Precision	Piggybacking	Complexity	Convergence time	GUI services	Network size	Sleep mode
RBS [19]	1.85 ± 1.28 μs	N/A	High	N/A	No	2–20 Nodes	Yes
Romer [56]	3 ms	Yes	Low	N/A	No	Unknown	Yes
Mock et al. [49]	150 μs	No	High	Low	No	Unknown	No
Ganeriwal et al. [25]	16.9 μs	No	Low	Unknown	No	150–300 Nodes	Yes
Ping [54]	32 μs	Yes	Low	High (multi-hop)	Yes	Unknown	No
PalChaudhuri et al. [53]	Unknown	Unknown	High	N/A	No	Unknown	Yes
Sichitiu et al. [58]	945 μs	No	Low	High (multi-hop)	No	N/A	Yes
Time Diffusion Protocol [62]	100 μs	No	High	High (multi-hop)	No	200 Nodes	Yes
Asynchronous diffusion [42]	Unknown	No	High	High (multi-hop)	No	200–400 Nodes	Yes

In our evaluations, we consider both the asymptotic behavior of a protocol's computation time and its memory requirements, relative to the number of nodes being synchronized. Even though a protocol's computational requirements might be linear in the number of nodes synchronizing with each other, the protocol may still be impractical if these requirements exceed physical node resources. RBS [19], the protocol by Mock et al. [49], asynchronous diffusion [42], and TDP [62] have higher computational and storage complexity compared to Romer's protocol [56], network-wide time synchronization [25], and the protocol by Sichitiu and Veerarittiphan [58].

Convergence time. Convergence time is the total time required to synchronize a network. A protocol that requires a large number of message exchanges per synchronization will result in a longer convergence time. As discussed earlier, reducing message complexity is vital to the cost-efficiency of a synchronization protocol for sensor networks. Since convergence time is directly proportional to message complexity and bandwidth use, reducing the convergence time is also an important factor in wireless networks.

RBS [19], Romer's protocol [56], and the protocol by PalChaudhuri et al. [53] do not emphasize low convergence time because they are based on reactive routing. In these protocols, synchronization is performed relatively infrequently, when an event of interest occurs in the net. Thus, convergence time and message complexity are of less critical importance in protocols that use reactive routing. The protocol by Mock et al. [49] requires a low convergence time because only one message is required per synchronization round. Other protocols [54,58] can tolerate high convergence times for multi-hop networks.

GUI services. Graphic User Interface (GUI) services can present, in a clear and comprehensible manner, meaningful results achieved by the network to an end user. Since some wireless networks require on-line monitoring by human users, we included the existence of GUI services in our performance comparisons. Only Ping's protocol [54] provides such services to the application and higher-level kernel modules. Two services are specifically defined: (1) time reading and (2) synchronized network event scheduling. These services allow a user to schedule a synchronized event over a network by providing an interface to the user's timeframe.

Network size. Some authors have conducted empirical evaluations of synchronization protocols on actual sensor networks. Although this information is not available for most of the protocols that we studied, the network-wide time synchronization protocol of Ganeriwal et al. [25] is noteworthy in this regard. This protocol was found to handle neighborhoods with up to 300 nodes.

Compatibility with sleep mode. The ability of a node to be in low-power (sleep) mode can be critical to meeting the node's energy requirements. The key idea underlying *sleep mode* is that nodes must be synchronized and active only when the application demands it. RBS [19] highlights this feature by way of post-facto synchronization. Other protocols [24,25,56] support this feature as well.

Table 5 shows the performance of the protocols relative to the quantitative criteria. The RBS protocol [19] yields excellent accuracy results even in the case of networks with modestly-equipped sensor nodes. The accuracy reached by the protocol is truly outstanding, in the order of few microseconds, for a network of nodes (i.e., the Berkeley motes) with severely limited computational and energy resources. The convergence time and message complexity of the protocol are relatively high, of the order of $O(m \cdot n^2)$ messages for m synchronization rounds involving n nodes within single-hop reach of each other. However, the CPU load and storage requirements of the protocol are modest, making RBS appropriate for even the simplest of today's sensor nodes. Moreover, the lack of clock adjustments sharply reduces the energy needs of the protocol relative to other protocols. Romer's protocol [56] achieves reasonably good accuracy (in the order of few milliseconds) with low computational complexity. The advantages of continuous time synchronization [49] are high accuracy results and low convergence time; however, these results are obtained at the expense of computational complexity.

Network-wide Time Synchronization [24,25] is an excellent compromise among synchronization accuracy, computational complexity, and convergence time. While the accuracy results are in the order of tens of microseconds, low computational complexity and fast convergence time make this protocol quite attractive when higher accuracy is not required. An additional strength of this protocol is that the protocol has been tested on an actual sensor network containing 300 nodes. The Delay-Measurement Time Synchronization protocol has comparable accuracy results as Network-Wide Time Synchronization [54]. However, this result is obtained at the expense of a longer convergence time. Improvements to Network-Wide Time Synchronization were defined by Dai and Han [15]. Their method has yielded excellent accuracy results with lower message complexity than RBS.

The protocol by PalChaudhuri et al. [53] is interesting because it uses a probabilistic algorithm. While probabilistic algorithms hold considerable promise for clock synchronization in sensor networks, at this time it is unclear how these protocols will perform in practice. Finally, a protocol by Sichitiu and Veerarittiphan [58] achieves good accuracy results (less than one millisecond). This is quite impressive considering that the protocol also has low computational complexity. However, the convergence time of this protocol is quite high.

The asynchronous diffusion protocol of Li and Rus [42] and the time diffusion protocol of Su and Akyildiz [62] use an averaging method to adjust node clocks. These protocols are reasonably robust; however, the issue of clocks running backward must be suitably addressed before these protocols are implemented in practice.

5.2. Qualitative evaluation

We evaluate the protocols based on overall quality criteria. In contrast to Section 5.1 above, here we discuss the goals of each protocol and the extent to which we deem that the protocol succeeds in achieving those goals. While a quantitative study deals with parameters that help the reader fine-tune a synchronization protocol by providing a telescopic view, a qualitative study provides a broader and more general perspective. Table 6 compares the various protocols in terms of the following qualitative criteria.

Energy efficiency. Energy efficiency is an implicit requirement in most wireless networks. The extent to which this requirement must be enforced will vary depending on an application. For instance, in the case of sensor networks the requirement is quite strict, forcing nodes to sleep as frequently as possible and severely limiting the energy available for synchronization and other network tasks. The main reason behind this energy constraint is the small size of batteries in sensor nodes, which greatly limits the amount of energy that can be stored and produced (e.g., with solar cells).

An important tradeoff for wireless networks is between using the available energy for computing or for communicating. Pottie and Kaiser [55] have shown that, in the case of sensor networks using radio frequency transmitters and receivers, the energy required to transmit 1 bit over 100 m,

which is 3 joules, can be used to execute 3 million instructions. This finding makes it clear that communication is far more energy-intensive than computation. Elson and Romer [20] have stressed that in low-power radio networks, listening and sending and receiving messages requires much more energy than in a wired network. The CPU is also sparingly available because it is shut down often to conserve energy.

Accuracy. Accuracy is a measure of how well the time maintained within the network is true to the standard time. In other words, it is a measure of the precision of synchronization. A protocol with high accuracy thereby guarantees high precision. In the case of absolute precision, this means that the synchronized time in the network does not deviate much from an external standard (e.g., UTC or GPS). In the case of relative precision, this means that, when a set of synchronized nodes is considered, the maximum deviation of the clock of any node within the set is reasonably small.

Several protocols considered in this survey are highly accurate. (See, e.g., [19,58].) Probabilistic synchronization [53] allows a user to choose a desired level of accuracy. The option of trading off accuracy for a lower complexity could be quite useful, depending on the network's load.

Scalability. Elson and Romer [20] state that the scope of a network is the geographic span of nodes that are synchronized and the completeness of coverage within that region. In general, the scope of a network can be expanded by increasing the number of nodes in the network. As the sensors are becoming cheaper, wireless sensor networks are becoming increasingly large, up to tens of thousands of nodes. Thus, synchronization protocols must be sufficiently scalable with respect to network size. Most protocols that handle sensor networks place scalability on top of their list of priorities.

Although most authors have not measured scalability in their experiments, Ganeriwal et al. [24,25] have used a network of 150–300 nodes to test scalability. In addition, Su and Akyildiz used a net with 200 nodes to test the scalability of TDP [62]. RBS [19] typically synchronizes 3–20 nodes in a neighborhood; however, this protocol works well even in much larger networks using gateways between neighborhoods. Protocols in which the synchronization error increases with the size of the network, such as Romer's protocol [56], achieve scalability at the expense of accuracy.

Overall complexity. The quantitative evaluation in the previous subsection distinguishes various complexity measures, including CPU load, storage requirements, and message complexity (i.e., convergence time). In this section, overall complexity is viewed as a combination of algorithmic complexity, overhead caused due to fault tolerance provisions, and communication overhead. Romer's protocol [56] is probably the best choice for a resource-restricted environment because of its low complexity. However, this method is not highly accurate. This is not surprising as highly accurate protocols usually incur high overall complexity.

Fault tolerance. Fault tolerance plays an important role because a wireless medium is rather

Table 6
Qualitative performance tabulation of synchronization protocols

Protocols	Qualitative performance comparison				
	Accuracy	Energy efficiency	Overall complexity	Scalability	Fault tolerance
RBS [19]	High	High	High	Good	No
Romer [56]	Average	High	Low	Poor	No
Mock et al. [49]	High	Low	Low	N/A	Yes
Ganeriwal et al. [25]	High	Average	Low	Good	Yes
Ping [54]	High	High	Low	Good	No
PalChaudhuri et al. [53]	Unknown	High	Low	Good	No
Sichitiu and Veerarittiphan [58]	High	High	Low	N/A	Yes
Time Diffusion Protocol [62]	High	Average	High	Good	Yes
Asynchronous diffusion [42]	Unknown	Low	High	N/A	Yes

error-prone. The poor reliability of message delivery in a wireless medium can have devastating effects on synchronization protocols because synchronization requires message exchanges.

Some fault-tolerant protocols [24,49,58] address message loss to some extent, but others have not addressed this issue. Consequently, it is unclear how sensitive their protocols are to message loss. This is somewhat troublesome because handling message loss can result in significant overheads and performance degradation during synchronization.

6. Conclusions

With increasing frequency, attention has been focused on wireless sensor networks because of their wide applicability to a diverse range of application areas. Among the many difficulties in designing and building such networks, a central challenge is providing clock synchronization among the sensor nodes. Providing a common time axis is necessary for a large number of sensor applications because the data they report has to be meaningfully fused to draw coherent inferences about the environment being sensed.

Traditional clock synchronization protocols for wired networks cannot be used because wireless sensor-network protocols require the ability to adapt dynamically, the ability to handle sensor mobility, and scalability. The sensors themselves are heavily resource-constrained because of limited battery power. Furthermore, they need to operate in highly lossy and unreliable environments. As a result, several clock synchronization protocols for wireless sensor networks have been designed in the recent past.

This paper presented a survey and analysis of existing clock synchronization protocols for wireless sensor networks, based on a variety of factors including precision, accuracy, cost, and complexity. The design considerations presented here will help the designer in building a successful clock synchronization scheme, best tailored to his application. Specifically, the detailed analysis of the various options and possible solutions for each of the factors involved will guide the designer in integrating various solution features to create a successful clock synchronization scheme for the application. Finally, the survey will be a helpful benchmark for designers to compare and contrast their results with the protocols that are widely in use.

References

[1] I.F. Akyildiz, O. Akan, C. Chen, J. Fang, W. Su, InterPlaNetary Internet: state-of-the-art and research challenges, Computer Networks 43 (2) (2003) 75–112.
[2] I.F. Akyildiz, W. Su, Y. Sankarasubramanian, E. Cayirci, A survey on sensor networks, IEEE Communications Magazine 40 (8) (2002) 102–114.
[3] I. Akyildiz, W. Su, Y. Sankarasubramanian, E. Cayirci, Wireless sensor networks: a survey, Computer Networks 38 (4) (2002) 393–422.
[4] K. Arvind, Probabilistic clock synchronization in distributed systems, IEEE Transactions on Parallel and Distributed Systems 5 (5) (1994) 474–487.
[5] S. Burleigh, V. Cerf, R. Durst, K. Fall, A. Hooke, K. Scott, H. Weiss, The InterPlaNetary Internet: a communications infrastructure for Mars exploration, in: 53rd International Astronautical Congress, The World Space Congress, October 2002.
[6] CENS: Center for Embedded Networked Sensing. Available at <http://www.cens.ucla.edu/index.html>.
[7] CENS: Monitoring of marine microorganisms. Available at <http://www.cens.ucla.edu/Research/Applications/momm.htm>.
[8] CENS: Seismic monitoring and structural response. Available at <http://www.cens.ucla.edu/Research/Applications/seismicmonitor.htm>.
[9] Chart on the Web. Maryland Department of Transportation. Available at <http://www.chart.state.md.us/>.
[10] J.-C. Chen, K.M. Sivalingam, P. Agrawal, S. Kishore, A comparison of MAC protocols for wireless local networks based on battery power consumption, in: IEEE Infocom'98, San Francisco, USA, March 1998, pp. 150–157.
[11] F. Cristian, Probabilistic clock synchronization, Distributed Computing 3 (1989) 148–153.
[12] F. Cristian, C. Fetzer, The timed asynchronous distributed system model, IEEE Transactions on Parallel and Distributed Systems 10 (6) (1999) 642–657.
[13] D. Culler, W. Hong (Eds.), Introduction, Communications of the ACM 47 (6) (2004) 30–34.
[14] D. Culler, M. Srivastava, D. Estrin (Eds.), Guest Editors' Introduction: Overview of Sensor Networks, IEEE Computer 37 (8) (2004) 41–49.
[15] H. Dai, R. Han, TSync: a lightweight bidirectional time synchronization service for wireless sensor networks, ACM SIGMOBILE Mobile Computing and Communications Review 8 (1) (2004) 125–139.

[16] Distributed Surveillance Sensor Network. ONR SPAWAR Systems Center, San Diego. Available from <http://www.spawar.navy.mil/robots/undersea/dssn/dssn.html>.
[17] A. Dobra, M. Garofalakis, J. Gehrke, R. Rastogi, Processing complex aggregate queries over data streams, in: Proceedings of SIGMOD International Conference on Data Management, 2002, pp. 61–72.
[18] D. Dolev, J.Y. Halpern, B. Simons, R. Strong, Dynamic fault-tolerant clock synchronization, Journal of the ACM 42 (1) (1985) 143–185.
[19] J. Elson, L. Girod, D. Estrin, Fine-grained network time synchronization using reference broadcasts, in: Proceedings of Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002), vol. 36, 2002, pp. 147–163.
[20] J. Elson, K. Romer, Wireless sensor networks: a new regime for time synchronization, in: Proceedings of First Workshop on Hot Topics In Networks (HotNets-I), Princeton, New Jersey, October 2002.
[21] R.C. Shah, J. Rabaey, Energy aware routing for low energy ad-hoc sensor networks, IEEE Wireless Communication and Networking Conference (WCNC), March 2002, pp. 350–355.
[22] D. Estrin, D. Culler, K. Pister, G. Sukhatme, Connecting the physical world with pervasive networks, IEEE Pervasive Computing 1 (1) (2002) 59–69.
[23] D. Estrin, R. Govindan, J. Heidemann, S. Kumar, Next century challenges: scalable coordination in sensor networks, in: Proceedings of Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99), August 1999, pp. 263–270.
[24] S. Ganeriwal, R. Kumar, S. Adlakha, M. Srivastava, Network-wide time synchronization in sensor networks, Technical Report, Networked and Embedded Systems Lab, Electronic Engineering Department, UCLA, 2003.
[25] S. Ganeriwal, R. Kumar, M. Srivastava, Timing-Sync protocol for sensor networks, in: Proceedings of First International Conference on Embedded Networked Sensor Systems, Los Angeles, California, November 2003.
[26] L. Girod, V. Bychkovskiy, J. Elson, D. Estrin, Locating tiny sensors in time and space: a case study, in: Proceedings of International Conference on Computer Design (ICCD 2002), September 2002.
[27] D.J. Goodman, R.A. Valenzuela, K.T. Gayliard, B. Ramamurthi, Packet reservation multiple access for local wireless communications, IEEE Transactions on Communications 37 (1989) 885–890.
[28] R. Gusella, S. Zatti, TEMPO—A network time controller for a distributed Berkeley UNIX system, IEEE Distributed Processing Technical Committee Newsletter 6 (S12-2) (1984) 7–14.
[29] J. Hill, M. Horton, R. Kling, L. Krishnamurthy, The platforms enabling wireless, sensor networks, Communications of the ACM 47 (6) (2004) 41–46.
[30] A.-S. Hu, S.D. Servetto, Asymptotically optimal time synchronization in dense sensor networks, in: Proceedings of 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA '03), San Diego, California, September 2003, pp. 1–10.
[31] IEEE, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Std. 802.11, November 1997.
[32] The Intel Mote. Intel Corporation. Available at <http://www.intel.com/research/exploratory/motes.htm>.
[33] P. Johnson, et al., Remote continuous physiological monitoring in the home, Journal of Telemedicine Telecare 2 (2) (1996) 107–113.
[34] C.E. Jones, K.M. Sivalingam, P. Agrawal, J. Cheng, A survey of energy efficient protocols for wireless networks, Wireless Networks 7 (4) (2001) 343–358.
[35] J.M. Kahn, R.H. Katz, K.S.J. Pister, Next century challenges: mobile networking for Smart Dust, in: Proceedings of Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking, 1999, pp. 271–278.
[36] M.J. Karol, Z. Liu, K.Y. Eng, An efficient demand-assignment multiple access protocol for wireless packet (ATM) networks, ACM/Baltzer Wireless Networks 1 (3) (1995) 267–279.
[37] A.D. Kshemkalyani, The power of logical clock abstractions, Distributed Computing 17 (2) (2004) 131–150.
[38] L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of the ACM 21 (7) (1978) 558–565.
[39] L. Lemmerman, et al. Earth science vision: platform technology challenges. in: Proceedings of International Geoscience and Remote Sensing Symposium (IGARSS 2001), Sydney, Australia, 2001.
[40] M.D. Lemmon, J. Ganguly, L. Xia, Model-based clock synchronization in networks with drifting clocks, in: Proceedings of 2000 Pacific Rim International Symposium on Dependable Computing, December 2000.
[41] Q. Li, M. DeRosa, D. Rus, Distributed algorithms for guiding navigation across a sensor network. in: Proceedings of Ninth Annual International Conference on Mobile Computing and Networking, September 2003, pp. 313–326.
[42] Q. Li, D. Rus, Global clock synchronization in sensor networks, in: Proceedings of IEEE Conference on Computer Communications (INFOCOM 2004), vol. 1, Hong Kong, China, March 2004, pp. 564–574.
[43] B. Liskov, Practical uses of synchronized clocks in distributed systems, in: Proceedings of Tenth Annual ACM Symposium on Principles of Distributed Computing, August 1991, pp. 1–9.
[44] S. Madden, R. Szewczyk, M. Franklin, D. Culler, Supporting aggregate queries over ad-hoc wireless sensor networks, in: Proceedings of Workshop on Mobile Computing Systems and Applications, 2002.
[45] L. Meier, P. Blum, L. Thiele, Internal synchronization of drift-constrained clocks in ad-hoc sensor networks, in: Proceedings of 5th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '04), Roppongi Hills, Japan, May 2004, pp. 90–97.

[46] D.L. Mills, Network time protocol (version 3): specification, implementation, and analysis, Technical Report, Network Information Center, SRI International, Menlo Park, CA, March 1992.
[47] D.L. Mills, Modelling and analysis of computer network clocks, Technical Report, 92-5-2, Electrical Engineering Department, University of Delaware, May 1992.
[48] D.L. Mills, Internet time synchronization: the network time protocol, IEEE Transactions of Communications 39 (10) (1991) 1482–1493.
[49] M. Mock, R. Frings, E. Nett, S. Trikaliotis, Continuous clock synchronization in wireless real-time applications, in: Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS-00), October 2000, pp. 125–133.
[50] S.B. Moon, P. Skelly, D. Towsley, Estimation and removal of clock skew from network delay measurements, Proceedings of IEEE INFOCOM, vol. 1, 1999, pp. 227–234.
[51] A. Olson, K. Shin, Fault-tolerant clock synchronization in large multicomputer systems, IEEE Transactions on Parallel and Distributed Computing 5 (9) (1994) 912–923.
[52] D. Raychaudhuri, N.D. Wilson, ATM-based transport architecture for multi-services wireless personal communication networks, IEEE Journal on Selected Areas in Communications 12 (1994) 1401–1414.
[53] S. PalChaudhuri, A. Saha, D.B. Johnson, Probabilistic clock synchronization service in sensor networks, Technical Report TR 03-418, Department of Computer Science, Rice University, 2003.
[54] S. Ping, Delay measurement time synchronization for wireless sensor networks, Intel Research, IRB-TR-03-013, June 2003.
[55] G. Pottie, W. Kaiser, Wireless integrated network sensors, Communications of the ACM 43 (5) (2000) 51–58.
[56] K. Romer, Time synchronization in ad hoc networks, in: Proceedings of ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01), October 2001, pp. 173–182.
[57] M. Ryu, S. Hong, Revisiting clock synchronization problems: static and dynamic constraint transformations for correct timing enforcement, Technical Report No. SNU-EE-TR-1998-3, Seoul National University, South Korea, September 1998.
[58] M.L. Sichitiu, C. Veerarittiphan, Simple, accurate time synchronization for wireless sensor networks, in: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2003), 2003, pp. 1266–1273.
[59] S. Singh, C.S. Raghavendra, PAMAS: power aware multi-access protocol with signaling for ad hoc networks, Distributed Computing 6 (1993) 211–219.
[60] The sensor web project, NASA Jet Propulsion Laboratory, Available from <http://sensorwebs.jpl.nasa.gov>.
[61] SmartDust, Autonomous sensing and communication in a cubic millimeter, Available at <http://robotics.eecs.berkeley.edu/~pister/SmartDust/>.
[62] W. Su, I. Akyildiz, Time-diffusion synchronization protocols for sensor networks, IEEE/ACM Transactions on Networking, in press.
[63] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, D. Estrin, Habitat monitoring with sensor networks, Communications of the ACM 47 (6) (2004) 34–40.
[64] S. Tilak, N.B. Abu-Ghazaleh, W. Heinzelman, Taxonomy of wireless micro-sensor network models, ACM Mobile Computing and Communications Review 6 (2) (2002) 28–36.
[65] TinyOS. An operating system for networked sensors, Available at <http://www.tinyos.net>.
[66] M. Tubaishat, S. Madria, Sensor networks: an overview, IEEE Potentials 22 (2) (2003) 20–23.
[67] D. Tulone, Resource-efficient time estimation for wireless sensor networks, in: S. Basagni, C.A. Phillips (Eds.), Proceedings of DIALM-POMC Workshop on Foundations of Mobile Computing, October 2004, pp. 52–59, Philadelphia, Pennsylvania. Available at <http://www.informatik.uni-trier.de//~ley/db/conf/dialm/dialm2004.html>.
[68] J. van Greunen, J. Rabaey, Lightweight time synchronization for sensor networks, in: Proc. 2nd ACM Int. Workshop on Wireless Sensor Networks and Applications (WSNA '03), San Diego, California, September 2003, pp. 11–19.
[69] R. Want, A. Hopper, V. Falcäo, J. Gibbons, The active badge location system, ACM Transactions on Information Systems 10 (1) (1992) 91–102.
[70] J. Werb, C. Lanzl, Designing a positioning system for finding things and people indoors, IEEE Spectrum 35 (9) (1998) 71–78.
[71] A. Woo, S. Madden, R. Govindan, Networking support for query processing in sensor networks, Communications of the ACM 47 (6) (2004) 47–52.
[72] Available at <http://deepspace.jpl.nasa.gov/dsn>.
[73] Available at <http://marsnet.jpl.nasa.gov>.
[74] Y. Yao, J. Gehrke, Query processing in sensor networks, in: Proceedings of the First Biennial Conference on Innovative Data Systems Research, Asilomar, California, January 2003.
[75] W. Ye, J. Heidemann, D. Estrin, An energy-efficient MAC protocol for wireless sensor networks, in: Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002), June, 2002, pp. 1567–1576.
[76] X.Yu, K. Niyogi, S. Mehrotra, N. Venkatasubramanian, Adaptive middleware for distributed sensor environments. IEEE Distributed Systems Online, May 2003.
[77] W. Yuan, S. Krishnamurthy, S. Tripathi, Synchronization of multiple levels of data fusion in wireless sensor networks, in: Proceedings of IEEE Global Telecommunications Conference (Globecom), pp. 221–225, 2003.
[78] J. Zhao, R. Govindan, D. Estrin, Computing aggregates for monitoring wireless sensor networks, in: Proceedings of IEEE Workshop on Sensor Network Protocols and Applications (SANPA), May 2003, pp. 139–148.

Bharath Sundararaman was born and raised in Chennai, India and currently lives in Palatine, Illinois. Bharath received a Bachelor of Engineering degree in Computer Science from Madras University, India in 2000 and a Master of Sciences degree in Computer Science from the University of Illinois at Chicago in 2003. He currently works as a Senior Software Engineer for Starthis Inc., a provider of automation middleware solutions in Arlington Heights, Illinois.

Ugo Buy is an Associate Professor in the Department of Computer Science of the University of Illinois at Chicago. He obtained his MS and PhD degrees in Computer Science from the University of Massachusetts at Amherst in 1983 and 1990, respectively. From 1983 to 1986 he was a Senior Software Engineer with the Digital Equipment Corporation in Hudson, Massachusetts. He has been on the UIC faculty since 1990.

His research interests are in the area of software engineering. In the past, he has been involved in several NSF-sponsored projects whose common goal was the definition of techniques and tools for the static analysis of concurrent and real-time software. Using a variety of program representations (e.g., finite-state automata and Petri nets), he investigated several techniques for promoting the reliability of concurrent and real-time software.

He is currently investigating techniques for the automatic synthesis of supervisory controllers for discrete manufacturing systems. The controllers must enforce traditional safety and timing properties of manufacturing plants. An additional research interest is the exploration of new computing applications based on wireless sensor networks.

Ajay Kshemkalyani is an Associate Professor of Computer Science at the University of Illinois at Chicago since 2000. He previously spent several years at IBM Research Triangle Park working on various aspects of computer networks. Ajay Kshemkalyani received a Ph.D. and an M.S. in Computer and Information Science from The Ohio State University in 1991 and 1988, respectively, and a B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Bombay, in 1987. His current research interests include computer networks, distributed computing, algorithms, and concurrent systems. He is a recipient of the National Science Foundation's CAREER Award. He is a member of the ACM and a senior member of the IEEE.