The Flooding Time Synchronization Protocol

Miklós Maróti Branislav Kusy Gyula Simon Ákos Lédeczi
Institute for Software Integrated Systems
Vanderbilt University
2015 Terrace Place, Nashville, TN 37203, USA
Phone: (+1) 615-343-7472
e-mail: {miklos.maroti, branislav.kusy, gyula.simon, akos.ledeczi}@vanderbilt.edu

ABSTRACT
Wireless sensor network applications, similarly to other distributed systems, often require a scalable time synchronization service enabling data consistency and coordination. This paper describes the Flooding Time Synchronization Protocol (FTSP), especially tailored for applications requiring stringent precision on resource limited wireless platforms. The proposed time synchronization protocol uses low communication bandwidth and it is robust against node and link failures. The FTSP achieves its robustness by utilizing periodic flooding of synchronization messages, and implicit dynamic topology update. The unique high precision performance is reached by utilizing MAC-layer time-stamping and comprehensive error compensation including clock skew estimation. The sources of delays and uncertainties in message transmission are analyzed in detail and techniques are presented to mitigate their effects. The FTSP was implemented on the Berkeley Mica2 platform and evaluated in a 60-node, multi-hop setup. The average per-hop synchronization error was in the one microsecond range, which is markedly better than that of the existing RBS and TPSN algorithms.

Categories and Subject Descriptors
C.2 [COMPUTER-COMMUNICATION NETWORKS]: Network Architecture and Design, Network Protocols.

General Terms
Algorithms, Design, Performance, Reliability, Experimentation.

Keywords
Sensor Networks, Time Synchronization, Clock Synchronization, Clock Drift, Multi-hop.

1. INTRODUCTION
The advances in micro electro-mechanical systems (MEMS) technology, in digital circuits design, integration and packaging, and in wireless communication are leading to smaller, cheaper and low-power sensing and computing devices. Cell phones and handheld computers already enjoy the increased popularity of the public. These trends point towards radically new systems of thousands or even millions of tiny computing devices interacting with the environment and communicating with each other. Research teams are working on incorporating sensing, processing and communication in a volume of less than one cubic millimeter [6], while devices of the size of a coin built from off-the-shelf components are commercially available already. The UC Berkeley Mica2 and Mica2Dot motes are popular research platforms of this emerging technology [19].

Complex networks built from thousands of such devices are expected to affect many aspects of our lives. The potential applications of wireless sensor networks (WSN) include:

- Monitoring applications: Non-intrusive and non-disruptive environmental monitoring helps biologists to study sensitive wildlife habitats and people with certain medical conditions can receive constant monitoring through sensors [12]. Sensor networks monitor the structural health of the Golden Gate Bridge in San Francisco and the microclimates on Great Duck Island, Maine [9].
- Mobile commerce, inventory management: by measuring continuously changing conditions, WSN will influence the movement of commodities to locations where the need exists.
- Smart office, kindergarten: systems containing wireless sensors will be an integral part of our office space. They would improve the education process by tailoring it to the individual needs of a child [14], adapt to context, and coordinate activities of multiple children.
- Military applications: potential applications include surveillance, target tracking [15], countersniper systems [10] or battlefield monitoring that propagates information to the soldiers and vehicles involved in combat.

WSN are large-scale distributed systems, yet their unique characteristics, especially the severe resource constraints, require the reevaluation of traditional distributed algorithms for problems once considered to be solved. One of the basic middleware services of sensor networks is time synchronization. Time synchronization is required for consistent distributed sensing and control. Furthermore, common services in WSN, such as coordination, communication, security, power management or distributed logging also depend on the existence of global time.

In this paper, we describe the Flooding Time Synchronization Protocol (FTSP) for WSN in detail. When designing the FTSP, our goals were to achieve network-wide time synchronization with error in the micro-second range and scalability up to

SenSys'04, November 3–5, 2004, Baltimore, Maryland, USA.

hundreds of nodes, while being robust to network topology changes and link and node failures. The proposed algorithm compensates for the relevant error sources by utilizing the concepts of MAC layer time-stamping [3], [7] and skew compensation with linear regression [2]. While these ideas have been utilized before, their unique combination and its effective implementation yield significantly better precision and somewhat lower communication overhead than existing approaches on the same platform. Finally, the implicit dynamic topology management of the FTSP provides rapid convergence and robustness.

The algorithm was implemented on Mica/Mica2 platforms running the TinyOS operating system [4]. A short overview of the target platform is given in Section 2. We offer a survey of existing time synchronization algorithms, emphasizing the algorithms utilizing similar ideas as FTSP, in Section 3. The possible sources of errors of radio message based synchronization are described and analyzed in Section 4. We describe the proposed FTSP algorithm in details and evaluate it based on a large scale experiment in Section 5. In Section 6, we compare FTSP to existing algorithms. Finally, an application using FTSP is described in Section 7 and we offer our conclusions and plans for further improvements in Section 8.

2. THE TARGET PLATFORM

One of the most widely used WSN platforms is the Berkeley Mica2 mote [19], [4]. The Mica2 mote has a 7.37 MHz processor, 4 kB of RAM, 128 kB of flash memory, 433 MHz wireless radio transceiver (38.4 kbps transfer rate, 500 feet maximum range), and is powered by two AA batteries. Pluggable sensor boards with temperature, light, magnetic and other sensors are available.

The Berkeley motes run the TinyOS operating system [4], [17], an open source, event driven and modular OS designed to be used with networked sensors. TinyOS handles task scheduling, radio communication with error detection, clocks and timers, ADC, I/O and EEPROM abstractions, and power management. Application developers can select a subset of the modules implementing these functionalities, extend or override them if necessary, and statically compile them into the final executable.

3. APPROACHES TO TIME SYNCHRONIZATION

Time synchronization algorithms providing a mechanism to synchronize the local clocks of the nodes in the network have been extensively studied in the past. The most widely adapted protocol used in the internet domain is the Network Time Protocol (NTP) devised by Mills [11]. The NTP clients synchronize their clocks to the NTP time servers with accuracy in the order of milliseconds by statistical analysis of the round-trip time. The time servers are synchronized by external time sources, typically using GPS. The NTP has been widely deployed and proved to be effective, secure and robust in the internet. In WSN, however, non-determinism in transmission time caused by the Media Access Channel (MAC) layer of the radio stack can introduce several hundreds of milliseconds delay at each hop. Therefore, without further adaptation, NTP is suitable only for WSN applications with low precision demands.

Two of the most prominent examples of existing time synchronization protocols developed for the wireless sensor network domain are the Reference Broadcast Synchronization (RBS) algorithm [2] and the Timing-sync Protocol for Sensor Networks (TPSN) [3].

In the RBS, a reference message is broadcasted. The receivers record their local time when receiving the reference broadcast and exchange the recorded times with each other. The main advantage of RBS is that it eliminates transmitter-side non-determinism. The disadvantage of the approach is that additional message exchange is necessary to communicate the local time-stamps between the nodes. To our best knowledge the algorithm has not been extended to large multi-hop networks.

The TPSN algorithm first creates a spanning tree of the network and then performs pairwise synchronization along the edges. Each node gets synchronized by exchanging two synchronization messages with its reference node one level higher in the hierarchy. The TPSN achieves two times better performance than RBS by time-stamping the radio messages in the Medium Access Control (MAC) layer of the radio stack [3] and by relying on a two-way message exchange. The shortcoming of TPSN is that it does not estimate the clock drift of nodes, which limits its accuracy, and does not handle dynamic topology changes.

4. UNCERTAINTIES IN RADIO MESSAGE DELIVERY

Non-deterministic delays in the radio message delivery in WSN can be magnitudes larger than the required precision of time-synchronization. Therefore, these delays need to be carefully analyzed and compensated for. We shall use the following decomposition of the sources of the message delivery delays first introduced by Kopetz and Ochsenreiter [7], [8] and later extended in [3] and [5].

(1) *Send Time*—time used to assemble the message and issue the send request to the MAC layer on the transmitter side. Depending on the system call overhead of the operating system and on the current processor load, the send time is nondeterministic and can be as high as hundreds of milliseconds.

(2) *Access Time*—delay incurred waiting for access to the transmit channel up to the point when transmission begins. The access time is the least deterministic part of the message delivery in WSN varying from milliseconds up to seconds depending on the current network traffic.

(3) *Transmission Time*—the time it takes for the sender to transmit the message. This time is in the order of tens of milliseconds depending on the length of the message and the speed of the radio.

(4) *Propagation Time*—the time it takes for the message to transmit from sender to receiver once it has left the sender. The propagation time is highly deterministic in WSN and it depends only on the distance between the two nodes. This time is less than one microsecond (for ranges under 300 meters).

(5) *Reception Time*—the time it takes for the receiver to receive the message. It is the same as the transmission time. The transmission and reception times overlap in WSN as pictured in Figure 1.

(6) *Receive Time*—time to process the incoming message and to notify the receiver application. Its characteristics are similar to that of send time.

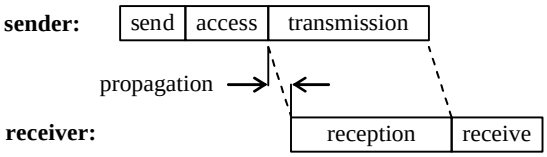

Figure 1. Decomposition of the message delivery delay over a wireless link.

The RBS approach completely eliminates the send and access times, and with minimal OS modifications it is also possible to remove the receive time uncertainty. This leaves the mostly deterministic propagation and reception time in wireless networks as the sole source of error. The main strength of RBS is its broad applicability to commodity hardware and existing software in sensor networks as it does not need access to the low levels of the operating system.

As the authors of the TPSN protocol observed, on typical WSN platforms, such as the Mica2 mote, one has direct access to the MAC layer, and message time-stamping can be performed during message transmission and reception. This immediately eliminates the same three main sources of uncertainties as in RBS. With a two-way handshake of synchronization messages the TPSN protocol eliminates the unknown propagation time as well.

Both the RBS and TPSN protocols suffer from the uncertainties of the overlapping transmission and reception times. To fully understand the constituents of this uncertainty we shall describe the message propagation in a typical wireless channel in more detail. We imagine an idealized point of the transmitted message, such as the end of a particular byte of the message. Then we follow the transmission of this idealized point through the software, hardware and physical levels of the wireless channel from the sender to the receiver.

First, the message is transferred to the radio chip piece by piece, usually in a byte oriented fashion. The radio chip signals the microcontroller that it is ready to obtain the next piece. The radio chip then encodes the pieces and generates an electromagnetic wave through the antenna. This wave propagates through space and the receiver's radio chip converts it back to binary representation. Then the radio chip on the receiver side signals the microcontroller that a new piece of data is ready and can be read through some protocol. Therefore, we have the following delivery delays of an idealized point of the message.

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

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

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

Some radio chips cannot capture the byte alignment of the transmitted message stream on the receiver side and the radio stack has to determine the bit offset of the message from the alignment of a known synchronization byte and then shift the message accordingly. Since the transmission time of the byte is a few hundred microseconds at 38.4 kbps, the delay caused by the incorrect byte alignment must be compensated for. This compensation is performed by the implementation of TPSN on the Mica2 platform, but it is not reported in [3].

(10) *Byte Alignment Time*—the delay incurred because of the different byte alignment of the sender and receiver. This time is deterministic and can be computed on the receiver side from the bit offset and the speed of the radio.

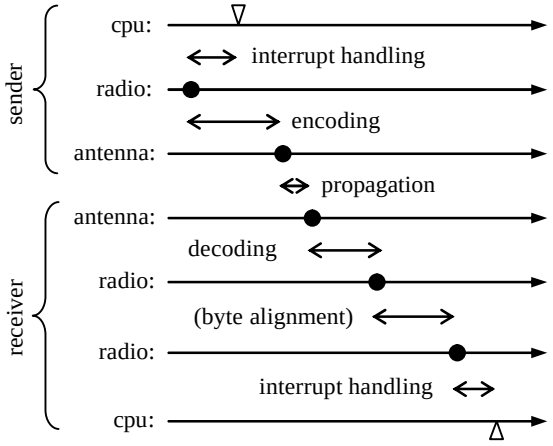

Figure 2. The timing of the transmission of an idealized point in the software (cpu), hardware (radio chip) and physical (antenna) layers of the sender and the receiver.

Figure 2 summarizes the decomposition of delivery delay of the idealized point of the message as it traverses over a wireless channel. Each line represents the time line of the layer as measured by an ideal clock. The dots represent the time instance when the idealized point of the message crosses the layers. The triangles on the first and last line represent the time when the cpu makes the time-stamps. Depending on the specific hardware the time stamp is usually recorded by the microcontroller when it handles the radio chip interrupts both on the sender and receiver sides. Alternatively, capture registers provided by some hardware can be employed to eliminate the interrupt handling time. We do not consider the effect of various coding techniques, such as Manchester or SECDEC coding or forward error-correction schemes, to the timing of message transmission. The codes used

in practice are block codes, and we can assume that the idealized point of the message is at a block boundary.

On the Mica2 platform, the interrupt handling time is typically around 5μs depending on the length of the code path between the start of interrupt handler and the part that records the local time. However, we observed interrupt handling times as high as 30μs. The sum of encoding and decoding times is between 110μs and 112μs. The byte alignment time is between 0μs (for bit offset 0) and 365μs (for bit offset 7). In contrast, the propagation time is under 1μs. Table 1 summarizes the magnitudes and distribution of the various delays in message transmissions.

Table 1. The sources of delays in message transmissions

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

Using our definitions we can properly express the sources of time-stamping errors of the RBS and TPSN algorithms. The RBS protocol is sensitive to the propagation, decoding and interrupt handling time differences between the two receivers. The main source of error here is the jitter in interrupt handling and decoding. The TPSN protocol is sensitive to the encoding, decoding and interrupt handling time differences between the sender and receiver. Note that although the propagation time has been eliminated, the encoding and decoding times are not because they might not be the same on the sender and receiver side. It is important to point out that both the RBS and TPSN protocols suffer from the two largest sources of uncertainty of MAC layer time-stamping: the jitter of interrupt handling and decoding time. On the other hand, as we will see in the next section, the FTSP time-stamping protocol effectively reduces all sources of time-stamping errors except for the propagation time.

5. FLOODING TIME SYNCHRONIZATION PROTOCOL

The goal of the FTSP is to achieve a network wide synchronization of the local clocks of the participating nodes. We assume that each node has a local clock exhibiting the typical timing errors of crystals and can communicate over an unreliable but error corrected wireless link to its neighbors.

The FTSP synchronizes the time of a sender to possibly multiple receivers utilizing a single radio message time-stamped at both the sender and the receiver sides. MAC layer time-stamping can eliminate many of the errors, as observed in [16] and [3]. However, accurate time-synchronization at discrete points in time is a partial solution only. Compensation for the clock drift of the nodes is inevitable to achieve high precision in-between synchronization points and to keep the communication overhead low. Linear regression is used in FTSP to compensate for clock drift as suggested in [2].

Typical WSN operate in areas larger than the broadcast range of a single node; therefore, the FTSP provides multi-hop synchronization. The root of the network—a single, dynamically (re)elected node—maintains the global time and all other nodes synchronize their clocks to that of the root. The nodes form an ad-hoc structure to transfer the global time from the root to all the nodes, as opposed to a fixed spanning-tree based approach proposed in [3]. This saves the initial phase of establishing the tree and is more robust against node and link failures and dynamic topology changes.

5.1 Time-stamping

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

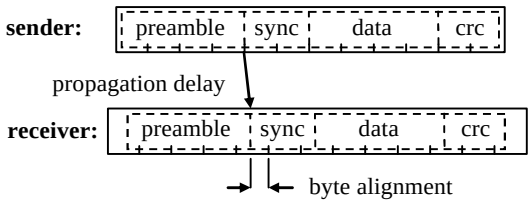

Figure 3. Data packets transmitted over the radio channel. Solid lines represent the bytes of the buffer and the dashed lines are the bytes of packets.

Message broadcast starts with the transmission of preamble bytes, followed by SYNC bytes, then with a message descriptor followed by the actual message data, and ends with CRC bytes. During the transmission of the preamble bytes the receiver radio synchronizes itself to the carrier frequency of the incoming signal. From the SYNC bytes the receiver can calculate the bit offset it needs to reassemble the message with the correct byte alignment. The message descriptor contains the target, the length of the data and other fields, such as the identifier of the application layer that

needs to be notified on the receiver side. The CRC bytes are used to verify that the message was not corrupted. The message layout is summarized in Figure 3.

The FTSP time-stamping effectively reduces the jitter of the interrupt handling and encoding/decoding times by recording multiple time stamps both on the sender and receiver sides. The time stamps are made at each byte boundary after the SYNC bytes as they are transmitted or received. First, these time stamps are normalized by subtracting an appropriate integer multiple of the nominal byte transmission time, the time it takes to transmit a byte. The jitter of interrupt handling time is mainly due to program sections disabling interrupts on the microcontroller for short amounts of time. This error is not Gaussian, but can be eliminated with high probability by taking the minimum of the normalized time stamps. The jitter of encoding and decoding time can be reduced by taking the average of these interrupt error corrected normalized time stamps. On the receiver side this final averaged time stamp must be further corrected by the byte alignment time that can be computed from the transmission speed and the bit offset. Note that, even though multiple time stamps are made, only the final error corrected time stamp is embedded into the message. The number of bytes put an upper limit on the achievable error correction using this technique. However, with only 6 time stamps, the time-stamping precision can be improved from tens of microseconds to 1.4µs on the Mica2 platform as measured by the following experiment.

Four motes were sending time-stamped messages to each other for 10 minutes, each with a 5-second sending period. The time-stamps were recorded both on the sender and receiver sides, and the pairwise clock offset and skew values were determined off-line with linear regression. The time-stamping error is the absolute value of the difference of the recorded receiver side time stamp and the linearly corrected sender side time-stamp. The average and maximum time-stamping errors were 1.4µs and 4.2µs, respectively. Since the FTSP time-stamping employs a single radio message, it does not and cannot compensate for the propagation delay. This is not a major limitation of the approach in typical WSN, however, as the propagation delay is less than 1µs for up to 300 meters.

5.2 Clock drift management

If the local clocks had the exact same frequency and, hence, the offset of the local times were constant, a single synchronization point would be sufficient to synchronize two nodes. However, the frequency differences of the crystals used in Mica2 motes introduce drifts up to 40µs per second. This would mandate continuous re-synchronization with a period of less than one second to keep the error in the micro-second range, which is a significant overhead in terms of bandwidth and energy consumption. Therefore, we need to estimate the drift of the receiver clock with respect to the sender clock.

The offset between the two clocks changes in a linear fashion provided the short term stability of the clocks is good. We verified the stability of the 7.37 MHz Mica2 clock by periodically sending a reference broadcast message that was received by two different motes. The two motes time-stamped the reference message using the FTSP time-stamping described in the previous section with their local time of arrival and reported the time-stamp. For each transmitted message the offset of the two reported time-stamps was calculated. The offsets were further examined: linear-regression was used to find the line *L* best approximating the dataset and the errors were analyzed. For a data point *(time,offset)* and the regression line *L*, the error is *offset-L(time)*. A one hour experiment produced the following results: the average value of the absolute errors was 0.95µs and the maximum absolute error was 4.32µs. The distribution of the errors, calculated off-line is shown in Figure 4. This off-line regression provides the best prediction that can possibly be achieved, provided the clocks can be considered stable during the experiment. Naturally this method cannot be used online; it is used here as a reference to evaluate online solutions.

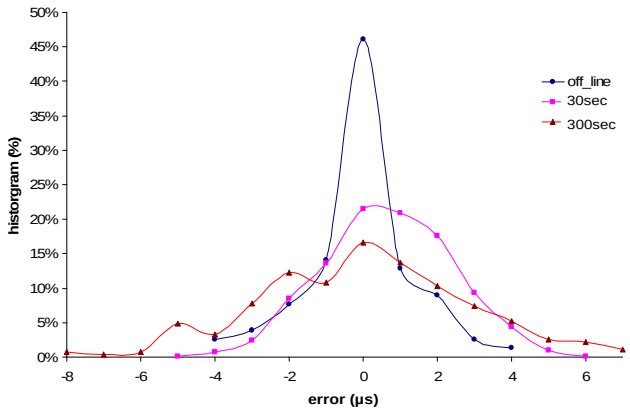

Figure 4. The distribution of the errors of linear-regression (LR): *off_line* refers to off-line LR, *30sec* refers to time sync interval P=30s, query interval 18s, and *300sec* refers to time sync interval P=300s, query interval 93s.

We need to identify the trend of the global time relative to the local time from the data points received in the past. Furthermore, only a limited number of data points can be stored due to the memory constraints of the platform. The following scenario was used to test our Mica2 implementation: mote *A* maintains the global time and sends synchronization messages to mote *B* with a period of *T*. Mote *B* estimates the skew and offset of its local clock from that of *A* using linear regression on the past 8 data points. A reference broadcaster sends a query message with period *t* and both *A* and *B* respond to this query by time-stamping its arrival with the global time and reporting it to the base station.

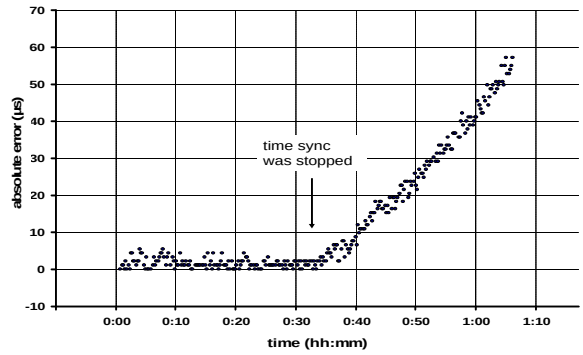

Figure 5. Time synchronization error between two motes. The time synchronization was stopped after 30 minutes. The initial small error of the skew estimate results in increasing synchronization error over time.

The linear regression prediction error is the difference between the global time given by *A* and the estimated global time given by *B*. Figure 4 shows the distribution of these prediction errors, for (a) *T*=30s, *t*=18s, and (b) *T*=300s, *t*=93s. The length of experiment (a) was 18 hours, the average absolute error was 1.48µs, and the maximum absolute error was 6.48µs. The length of experiment (b) was 8 hours, the average absolute error was 2.24µs and the maximum absolute error was 8.64µs.

An important design parameter is the required resynchronization interval to reach the desired precision. As shown in Figure 4, the 30s resynchronization interval gave slightly better results than 300s. To further evaluate the behavior of the skew compensation, another experiment was carried out, the results shown in Figure 5. This result shows that the resynchronization period, depending on the accuracy requirements, can go up to several minutes.

5.3 Multi-hop time synchronization

In practical WSN applications, the network radius is greater than one hop. If network-wide synchronization is required, the multi-hop FTSP protocol can be used, as described in this section. The only assumption the protocol makes is that every node in the network has a unique ID.

Nodes in multi-hop FTSP utilize *reference points* to perform synchronization. A reference point contains a pair of global and local time stamps where both of them refer to the same time instant, as described in Section 5.1. Reference points are generated by sending and receiving periodic broadcast messages, which are either transmitted by the *synchronization-root* (*root*, for short), or any *synchronized node* in the network. The root is a special node, elected and dynamically reelected by the network, to which the whole network is being synchronized. A node that is in the broadcast radius of the root can collect reference points directly from it. Nodes outside the broadcast radius of the root can gather reference points indirectly through other synchronized nodes that are located closer to the root. When a node collects enough consistent reference points, it estimates the offset and skew of its own local clock, as described in Section 5.2, and becomes synchronized. The newly synchronized node can then broadcast synchronization messages to other nodes in the network. In the following subsections the most important aspects of the protocol will be presented.

Synchronization Message Format: Each synchronization message contains three fields: the *timeStamp*, the *rootID*, and the *seqNum*. The *timeStamp* contains the global time estimate of the transmitter when the message was broadcasted. The *rootID* field contains the ID of the root, as known by the sender of the message. The *seqNum* is a sequence number set and incremented by the root when a new synchronization round is initiated. Other synchronized nodes insert the most recent (i.e. the largest) received *seqNum* into the synchronization messages they broadcast. This field is used to handle redundant synchronization messages.

Managing Redundant Information: Since all synchronized nodes periodically transmit synchronization messages, in a dense network a receiver may receive several messages from different nodes in a short time interval. Due to limited resources, an appropriate subset of the messages must be selected to create reference points. In the Mica2 implementation an eight-element regression table stores the selected reference points used to calculate the regression line, and, thus, the drift of the local clock.

To achieve more accurate offset and skew estimation using the limited amount of data that can be stored in the regression table, it is more beneficial to store reference points that are distributed over a longer period of time. To aid message filtering, each node maintains a *highestSeqNum* variable. Also each node has a *myRootID* variable, containing the root ID as known by the node. A received synchronization message is used to create a reference point only if the *rootID* field of the message is less than or equal to *myRootID* and the *seqNum* field is greater than *highestSeqNum* in the case when *rootID* = *myRootID*. The node's variables are updated after storing a reference point, according to the respective fields in the message. This message filtering protocol guarantees that only the first message arrived will be used in the reference table for each *rootID* and *seqNum* pair (i.e. one per round), providing reference points distributed over a longer time period for more accurate skew and offset estimation. The pseudo-code describing this protocol is presented at lines 3–9 in Figure 6 and 11–16 in Figure 7.

The root election problem: To perform global synchronization, obviously one and only one root is needed in the network. Since nodes may fail or the network can get disconnected, no dedicated node can play the role of the root. Thus a robust election process is needed to provide a root after startup, and also in case of root failure. FTSP utilizes a simple election process based on unique node IDs, as follows:

```
 1 event Radio.receive(TimeSyncMsg *msg)
 2 {
 3   if( msg->rootID < myRootID )
 4     myRootID = msg->rootID;
 5   else if( msg->rootID > myRootID
 6         || msg->seqNum <= highestSeqNum )
 7     return;
 8
 9   highestSeqNum = msg->seqNum;
10   if( myRootID < myID )
11     heartBeats = 0;
12
13   if( numEntries >= NUMENTRIES_LIMIT
14      && getError(msg) > TIME_ERROR_LIMIT )
15     clearRegressionTable();
16   else
17     addEntryAndEstimateDrift(msg);
18 }
```

Figure 6. The handling of new synchronization messages.

When a node does not receive new time synchronization messages for *ROOT_TIMEOUT* number of message broadcast periods, it declares itself to be the root (*myRootID := myID*). Thus after a *ROOT_TIMEOUT* period, there will be at least one, but possibly multiple roots in the network. Whenever a node receives a message with a *rootID* field smaller than its *myRootID* variable, it updates the variable according to the received *rootID* field. This mechanism ensures that roots with higher IDs give up their status and eventually there will be only one root—the node with the smallest ID—in the whole network. The pseudo-code describing

this root election protocol is given at lines 3–4, 10–11 in Figure 6 and 3–7 in Figure 7. The *heartBeats* variable contains the number of message broadcast periods since the last synchronization point was inserted into the regression table.

During the election process special care is taken to avoid inconsistencies and maintain the synchronized state of the network as much as possible. Only the root and synchronized nodes, those that have enough entries (at least *NUMENTRIES_LIMIT* many) in their linear regression table, transmit time synchronization messages (see lines 9–10 in Figure 7). If a node receives a new reference point that is in disagreement with previous estimates of the global time, then it clears its regression table (see lines 13–15 in Figure 6). Finally, a newly elected root preserves its estimated skew and offset parameters (does not clear its regression table) and broadcasts the global time accordingly. This way the network will not get out of synchronization during the reelection phase.

Another root related problem may arise if a new node is introduced to the network with a lower ID than the current root of the network. In this case, the new root eventually declares itself root, but again care is taken to provide a smooth transition. The new root does not start transmitting its own synchronization messages for the *ROOT_TIMEOUT* period, while it collects reference points to estimate its own skew and offset values. Thus, the global time broadcasted by the new root will be synchronized to the previous global time.

```
 1 event Timer.fired()
 2 {
 3   ++heartBeats;
 4
 5   if( myRootID != myID
 6       && heartBeats >= ROOT_TIMEOUT )
 7     myRootID = myID;
 8
 9   if( numEntries >= NUMENTRIES_LIMIT
10       || myRootID == myID ){
11     msg.rootID = myRootID;
12     msg.seqNum = highestSeqNum;
13     Radio.send(msg);
14
15     if( myRootID == myID )
16       ++highestSeqNum;
17   }
18 }
```

Figure 7. Periodic sending of synchronization messages.

Convergence properties: The convergence of the network to a globally synchronized state after a startup or failure depends on the speed of information propagation in the network. Since synchronization messages are broadcasted periodically, but the individual nodes are transmitting *asynchronously*, the speed of message propagation is limited.

Since node failures can be handled smoothly once the network is synchronized, it is more important to analyze the behavior of the system after startup. Denote the value of *NUMENTRIES_LIMIT* by *N*, the value of *ROOT_TIMEOUT* by *M*, the message broadcast period by *P*, and the radius of the network measured from the root by *R*. We assume that all links are reliable and the regression table is not cleared because of erroneous synchronization points (see lines 13–14 in Figure 6). There is no elected root in the network when the nodes are turned on. It takes *P*M* time for all nodes to declare itself the root (as none of them received synchronization messages), after which the election algorithm begins. To estimate the skew and offset of a local clock and to transmit synchronization messages, each node needs at least *N* entries in its regression table. The minimum and maximum times for the network to get synchronized to the node with the lowest node ID, once it starts transmitting synchronization messages, are *P*(N-1)*R* and *P*N*R*, respectively. Therefore, the total time to get synchronized is between *P*(M+(N-1)*R)* and *P*(M+N*R)*.

A synchronization period—the time when messages with the same *rootID* and *seqNum* are propagating in the network—lasts for at most *P*R* time, and often less because the nodes send synchronization messages asynchronously and have more than one neighbor. To analyze the convergence in the case when the root fails, denote by *R'* the radius of the network measured from the new root (the node with the second smallest node ID). After the old root fails, it takes at most *P*R* time for each node to receive messages of the last synchronization period and then an additional *P*M* waiting time to declare itself the root. At this point multiple nodes became the root, and a new reelection process begins. This lasts at most *P*R'* time because all nodes have full regression tables which do not get cleared when new synchronization messages arrive (see lines 13–15 in Figure 6). Thus, the total time of the reelection process is at most *P*(R+M+R')*.

The choice of the period *P* is a tradeoff between power consumption, accuracy, and speed of convergence: decreasing the period increases the number of messages sent in a certain time period but it also increases accuracy and allows faster convergence. Figure 4 gives information about the achievable accuracy per hop, using different *P* values.

5.4 Experimental data

The implementation of FTSP on the Mica and Mica2 platforms that was used to carry out the experiments described in this section is available on internet (see [18]). We tested the protocol focusing on the most problematic scenarios, such as switching off the root of the network, removing a substantial part of the nodes from the network, so that the remaining nodes still formed a connected network, and switching on a substantial number of the new nodes in the network.

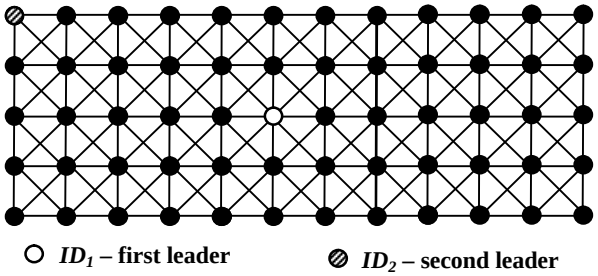

Figure 8. The layout and links of the experimental setup. Each node can only communicate with its (at most 8) neighbors.

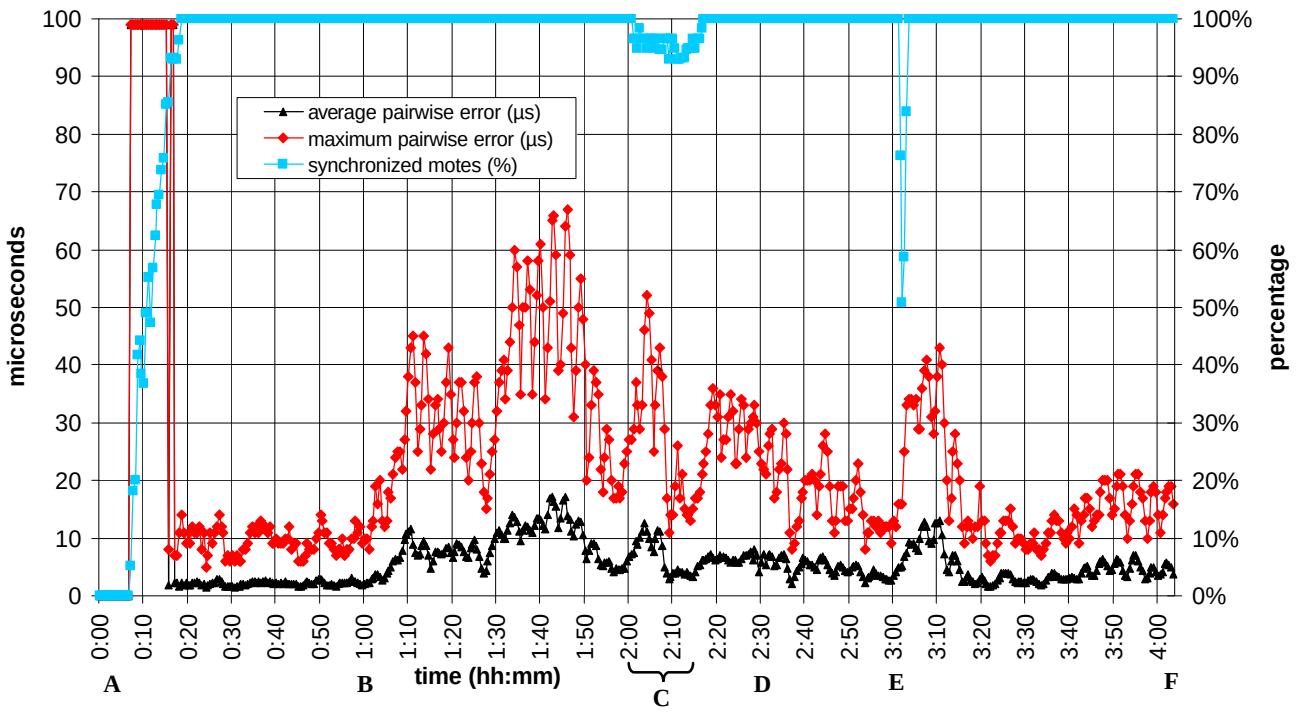

Figure 9. A 5x12 grid experiment shows the percentage of synchronized nodes, the maximum and average error (the maximum and average of the pairwise differences of the reported global times). The nodes were switched on at time A, the root ID_1 was switched off at B, randomly selected motes were reset during C, half of the motes were switched off at D, the same motes were switched back on at E, and the experiment ended at F.

The experiment scenario involves 60 Mica2 motes deployed in a 5x12 grid in such way that each mote can communicate with its neighbors only. Furthermore, the node with the smallest id (ID_1) is located in the middle of the network and the node with the second smallest id (ID_2) is at the edge of the network as shown in Figure 8. This means that ID_1 will become the first root of the network and ID_2 will replace ID_1 if and when it fails. The maximum hop distance between ID_1 and ID_2 represents the worst case scenario in the case the root ID_1 dies. The chosen topology also ensures that there exist enough motes at the distant levels (i.e. for the ID_2 root, there are at least 5 motes at each hop distance level, except for the hop distance 1).

Two other motes were used in the experiment, the reference broadcaster, and the base station. Their function was described in Section 5.2: the reference broadcaster queried the global time from all nodes in the network once per 30 seconds and the base station collected the responses to the query from all the nodes.

The topology of the 60 nodes network was enforced in software and, therefore, all the nodes could be placed within the radio range from the reference broadcaster and the base station. This way the base station and the broadcaster could talk directly to all 60 nodes and no multi-hop routing was necessary. The period of time re-synchronization was 30 seconds for all 60 nodes. The value of *NUMENTRIES_LIMIT* was 3, and *ROOT_TIMEOUT* was 6, that is, it took a node 6 times 30 seconds, i.e., 3 minutes to declare itself the root if it did not receive new synchronization messages. The experiment took about four hours with following scenario:

A: at 0:04 all motes were turned on;

B: at 1:00 the root with ID_1 was switched off, ID_2 becomes the new root, eventually;

C: between 2:00 and 2:15 randomly selected nodes were reset one by one with 30 second period;

D: at 2:30 the motes with odd node IDs were switched off (half of the nodes are removed);

E: at 3:01 the motes with odd node IDs were switched back on (100% new nodes were introduced);

F: at 4:02 the experiment was ended.

There were 60 nodes in the network, and typically less than 5% of them did not succeed to reply to the reference broadcaster due to radio collisions. The nodes reported back to the base station whether they were synchronized (i.e. had enough values in their regression table) and what the global time was at the arrival of the reference broadcast message.

For each reference broadcast round, we calculated the percentage of the motes that were synchronized out of those that replied, and we analyzed the time synchronization error by calculating the average of the pairwise differences of the reported global times, and the maximum difference of any two reported global times. We call these values the *average* and *maximum time synchronization error*, respectively. The resulting graph is shown in Figure 9.

An alternative definition of the accuracy of the FTPS could be the average difference between the estimated global time of each node and the real global time, i.e. the local time of the root. However,

when the root fails, this measure becomes undefined. Furthermore, typical WSN applications do not care about the absolute time as such. What is important is how well the nodes are synchronized to each other. Our definition of average error tries to capture this quality.

The nodes were switched on approximately at the same time (0:04), but during the next 3 minutes (till 0:07) no node was synchronized because none of them declared itself to be the root. Then many nodes timed out and became the roots of the network, which was the reason why the average and maximum synchronization errors soared for 10 minutes untill the election process has completed (0:17) and only a single root remained (ID_1). The number of synchronized nodes grew steadily during this period and the average and maximum errors became approximately 2μs and 10μs, respectively. Complete synchronization has been achieved in 14 minutes (at 0:18) as indicated by the percentage of synchronized motes reaching 100%. According to Section 5.3, in an idealized network the predicted minimum and maximum convergence times would be 9 and 12 minutes, respectively. In reality this took longer because there were radio collisions and some clock skew and offset estimates were erroneous.

When the root ID_1 was switched off, no impact on the network was immediately observable. What happened is that the global time had not been updated for a certain period of time until each node timed out and declared itself to be the root.

The election process again resulted in a single root (ID_2) eventually. However, the error stayed low during this time because nodes did not discard their old offset and skew estimates and the new root was broadcasting its estimation of the old global time. This caused deterioration of the maximum and average errors until all nodes calculated more accurate drift estimates based on the messages broadcasted by the new root. From Figure 9 one cannot determine when the network got synchronized to the second root, but from logged synchronization messages we determined that the election process finished at 1:06 (in 6 minutes). According to the formulas in Section 5.3 the election process lasts at most 11 minutes and 30 seconds in an ideal network.

In the last two parts of the experiment some of the nodes were removed and new ones were introduced (or reset). The impact of these operations on the average and maximum errors was minimal. We can observe that the number of synchronized nodes decreased whenever a new node was switched on because it takes some time for the new node to obtain enough data to get synchronized.

Before we switched off the first root, we had a 60-mote 6-hop network, that is, the maximal minimum distance from the root was 6, for approximately 50 minutes. The maximum average error was 3μs translating to a 0.5μs error per hop. The maximum error was less than 14μs overall or 2.3μs per hop.

After the simulated root failure at time *B*, we had a 59-mote 11-hop network. The average time synchronization error stayed below 17.2μs for the remainder of the experiment, despite the significant changes in the topology. If we divide the error by number of hops, we get the average error of 1.6μs per hop. The maximum time synchronization error was below 67μs which was observed only when the root was switched off. Switching off and introducing the new nodes did not introduce a significant time synchronization error.

6. COMPARISON TO PREVIOUS APPROACHES

In this section the pros and cons of the proposed FTSP are compared to those of previously known protocols. As reference, the RBS and TPSN algorithms were chosen, because (1) these time synchronization protocols were also developed with the special requirements of sensor networks in mind as opposed to other, more general algorithms, (2) actual experimental results are available for the same platforms (Mica/Mica2), and (3) ideas from these protocols were used and enhanced in FTSP.

The RBS approach time-stamps messages only on the receiver side; therefore, it eliminates the access and the send times. The published method in [2] does not compensate for byte alignment, but that could be easily incorporated. The main achievement of the RBS time-stamping of a reference broadcast is that it eliminates random delays on the sender side. However, time-stamping the radio messages in the low layers of the radio stack used in our method has practically the same effect and eliminates the jitter of interrupt handling and decoding times, as well.

The TPSN approach [3] eliminates the access time, byte alignment time and propagation time by making use of the implicit acknowledgments to transmit information back to the sender. This protocol gains an additional accuracy over RBS due to time-stamping the radio message multiple times and averaging these time-stamps. TPSN was implemented on the Mica platform and it would face certain implementation problems on later platforms. Unfortunately in the Mica2 platform implicit acknowledgments can not be effectively implemented because of long settling time of the radio chip when switching from the receiving mode to transmission mode. Another disadvantage of the TPSN protocol is that the two-way communication prohibits the use of message broadcasting, which results in higher communication load.

The accuracy of the RBS time-stamping reported by the authors is ~11μs on the Mica platform. Least square linear regression is used to account for the clock drifts which results in 7.4μs average error between two motes after a 60 second interval. The multi-hop scenario involves the local time transferring through the intermediary nodes. However, the function of the Berkeley motes was limited to providing wireless communication to PDAs (iPAQ) that were carrying out the time synchronization. Through private communication the authors of RBS indicated improved 5μs time-stamping precision on the Mica2 platform, mainly due to the higher bit-rate of the radio. The authors of TPSN algorithm implemented both TPSN and RBS on the Mica platform using a 4 MHz clock for time-stamping, and compared the precision of the two algorithms. The resulting average errors for a single hop case for two nodes are 16.9μs and 29.1μs for the TPSN and RBS algorithms, respectively [3].

The proposed FTSP algorithm uses a fine-grained clock, MAC-layer time-stamping with several jitter reducing techniques to achieve high precision. This approach eliminates the send, access, interrupt handling, encoding, decoding and receive time errors, but does not compensate for the propagation time. Multiple time-stamps with linear regression are used to estimate clock skew and offset. The average error of the algorithm for a single hop case

using two nodes was 1.48μs, according to measurements described in Section 5.2. In the multi-hop case, the average error was 3μs in a 6-hop network, resulting in a 0.5μs per hop accuracy.

The applied flood-based communication protocol in FTSP provides a very robust network, and still induces only small network traffic. The network hierarchy is maintained using the time synchronization messages, without additional message passing, as opposed to the solution in TPSN. FTSP also utilizes less network resources than either RBS or TPSN. If the resynchronization period is *T* seconds, then each node sends 1 message per *T* seconds in FTSP, 2 messages per *T* seconds in TPSN (1 message to parent and 1 response) and 1.5 message per *T* seconds in RBS (0.5 for a reference broadcast and 1 for a time stamp exchange message). Since FTSP does not rely on a fixed network hierarchy but updates it continuously, it supports network topology changes including mobile nodes. The robustness of the protocol was demonstrated by the harsh experiment described in Section 5.4. Unfortunately, no similar data is readily available for TPSN or RBS for comparison.

7. APPLICATIONS

The FTSP algorithm was excessively tested as a component of a countersniper application [10], [13]. The system utilized a network of Mica2 motes each of which was attached to a custom acoustic sensor board. The sensors measured both the muzzle blast and shock wave to accurately determine both the location of the shooter and the trajectory of the bullet. The basic idea is simple: using the arrival times of the acoustic events at different sensor positions, the shooter position can be accurately calculated using the speed of sound and the location of the sensors provided the clocks of the sensor nodes are precisely synchronized. Thus, the time synchronization protocol was a key element of the system.

The application, in addition to the FTSP, contained several services, such as message routing, data aggregation, remote configuration and debugging services, along with application-specific software components. A typical test scenario involved 50 to 60 motes distributed in an urban environment. The network was approximately 8 hops wide. The system was tested repeatedly for 4 to 8 hours of continuous operation. During testing some of the motes were switched off and on, the temperature and humidity of the environment changed drastically influencing the stability of the crystals. All nodes remained synchronized during these tests, but no other explicit time synchronization data was obtained. However, the overall performance of the countersniper system (~1 meter localization accuracy in 3D in an urban environment) and the fact that there was no performance degradation over time, clearly verified that the FTSP performed well.

8. CONCLUSION AND FURTHER IMPROVEMENTS

We have described the Flooding Time Synchronization Protocol for WSN. The protocol was implemented on the UCB Mica and Mica2 platforms running TinyOS. The precision of 1.5μs in the single hop scenario and the average precision of 0.5μs per hop in the multi-hop case were shown by providing experimental results. This performance is markedly better than those of other existing time synchronization approaches on the same platform.

The presented FTSP time-stamping protocol could be applied to existing multi-hop time synchronization protocols making those more precise. It differs from previous time-stamping algorithms in that it utilizes a single broadcasted message to establish synchronization points between the sender and receivers of the message, while eliminating most sources of synchronization errors, except for the propagation time. Because of its performance, minimal overhead and simplicity, it can serve as the most basic time synchronization primitive in other, not necessarily time synchronization protocols. The presented multi-hop FTSP achieves its performance and robustness by exploiting these good properties of the FTSP time-stamping protocol in the context of a dynamic leader-election algorithm, and combining it with clock drift compensation algorithms.

The FTSP was tested and its performance was verified in a real-world application. This is significant because the service had to operate not in isolation, but as part of a complex application where resource constraints as well as intended and unintended interactions between components can and usually do cause undesirable effects. Moreover, the system operated in the field for extended periods and not under laboratory conditions. This is a testimony to the robustness of the protocol and its implementation.

We plan to perform further experiments in networks with a magnitude larger number of nodes (thousands). An active research area is to improve the convergence of the multi-hop case by using two different broadcast periods in the protocol: a small period for an initial synchronization period (until all the nodes get synchronized) and a long period for the normal operation of the time synchronization protocol.

9. ACKNOWLEDGMENTS

The DARPA/IXO NEST program (F33615-01-C-1903) has supported the research described in this paper.

10. REFERENCES

[1] Elson, J. E. *Time Synchronization in Wireless Sensor Networks.* Ph.D. Thesis, University of California, Los Angeles 2003.

[2] Elson, J. E., Girod, L., and Estrin, D. *Fine-Grained Network Time Synchronization using Reference Broadcasts.* The Fifth Symposium on Operating Systems Design and Implementation (OSDI), p. 147–163, December 2002.

[3] Ganeriwal, S., Kumar, R., and Srivastava, M. B. *Timing-Sync Protocol for Sensor Networks.* The First ACM Conference on Embedded Networked Sensor System (SenSys), p. 138–149, November 2003.

[4] Hill, J., and Culler, D. Mica: *A Wireless Platform for Deeply Embedded Networks.* IEEE Micro archive, Volume 22, Issue 6, p. 12–24, November 2002.

[5] Horauer, M. et. al. *PSynUTC – Evaluation of a High Precision Time Synchronization Prototype System for Ethernet LANs*. 34th Annual Precise Time and Time Interval Meeting (PTTI), December 2002.

[6] Kahn, J. M., Katz, R. H., and Pister, K. S. J. *Mobile Networking for Smart Dust.* In Proceedings of the 5th annual

ACM/IEEE international conference on Mobile computing and networking, p. 271–278, August 1999.

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

[8] Kopetz, H., and Schwabl, W. *Global time in distributed real-time systems.* Technical Report 15/89, Technische Universitat Wien, 1989.

[9] Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., and Anderson, J. *Wireless Sensor Networks for Habitat Monitoring.* ACM International Workshop on Wireless Sensor Networks and Applications, p. 88–97, September 2002.

[10] Maróti, M., Simon, Gy., Lédeczi, Á., Sztipanovits, J. *Shooter Localization in Urban Terrain.* IEEE Computer **37**, no. 8, (2004) August, p. 60–61.

[11] Mills, D. L. *Internet Time Synchronization: The Network Time Protocol.* IEEE Transactions on Communications COM **39** no. 10, p. 1482–1493, October 1991.

[12] Schwiebert, L., Gupta, S., and Weinmann, J. *Research Challenges in Wireless Networks of Biomedical Sensors.* SIGMOBILE 2001, p. 151–165, July 2001.

[13] Simon, G. et al. *Sensor Network-Based Countersniper System*. The Second ACM Conference on Embedded Networked Sensor Systems (SenSys), November 2004.

[14] Srivastava, M., Muntz, R., and Potkonjak, M. *Smart Kindergarten: Sensor-based Wireless Networks for Smart Developmental Problem-solving Environments.* Proceedings of the ACM SIGMOBILE 7th Annual International Conference on Mobile Computing and Networking, p. 132–138, July 2001.

[15] Yang, H., and Sikdar, B. *A Protocol for Tracking Mobile Targets using Sensor Networks.* IEEE Workshop on Sensor Network Protocols and Applications, May 2003.

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

[17] TinyOS, http://webs.cs.berkeley.edu/tos/

[18] http://cvs.sourceforge.net/viewcvs.py/tinyos/minitasks/02/vu/tos/lib/TimeSync/

[19] Mica2 and Mica2Dot: http://www.xbow.com/Products/Wireless_Sensor_Networks.htm