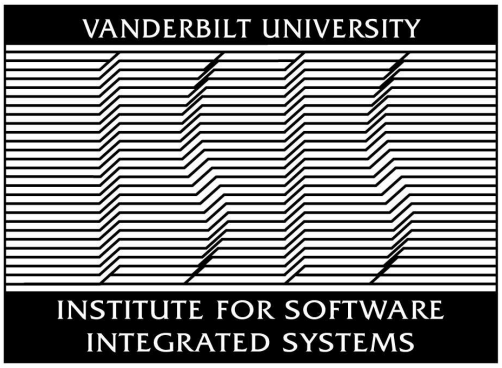

Institute for Software Integrated Systems
Vanderbilt University
Nashville Tennessee 37235

TECHNICAL REPORT

TR #: ISIS-04-501
Title: The Flooding Time Synchronization Protocol
Authors: Miklos Maroti, Branislav Kusy, Gyula Simon and Akos Ledeczi

The Flooding Time Synchronization Protocol

Miklos Maroti, Branislav Kusy, Gyula Simon and Akos Ledeczi
The Institute for Software Integrated Systems, Vanderbilt University
miklos.maroti@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 introduces the robust Flooding Time Synchronization Protocol (FTSP), especially tailored for applications requiring stringent precision on resource limited wireless platforms. The proposed time synchronization protocol utilizes low communication bandwidth, scales well for medium sized multi-hop networks, and is robust against topology changes and node failures. The FTSP achieves its robustness by utilizing periodic radio broadcast of synchronization messages, and implicit dynamic topology update. The unique high precision performance is reached by utilizing MAC-layer time-stamping, comprehensive error compensation, including linear regression, which reduces time skew and keeps network traffic overhead low. The performance of the FTSP, implemented on Berkeley MICA2 platform, was evaluated in a multi-hop experiment. The average network-wide synchronization error was in the microsecond range, which is approximately a magnitude lower than that of the existing RBS and TPSN algorithms. The protocol was further validated as part of our counter-sniper system that was field tested in a US military facility.*

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 lower 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 [9], 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 [10].

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]. The Golden Gate Bridge in San Francisco is monitored for structural health and sensor networks monitor the microclimates on Great Duck Island, Maine which is the habitat of Leach's Storm Petrel [11].
- *Mobile commerce, inventory management:* measuring continuously changing conditions, WSN will influence the movement of commodities to the locations where a need exists.
- *Smart office, kindergarten:* the systems containing wireless sensors can be part of our office space; the education process can be tailored to the individual needs of a child [13], adapt to context, or coordinate activities of multiple children.
- *Military applications:* potential applications include surveillance, target tracking [14], counter-sniper systems or battlefield monitoring that propagates information to the soldiers and vehicles involved in combat.

The unique characteristics of the sensor network domain demand the reevaluation of algorithms well established over a long period of time, and the design of new solutions for problems once considered to be solved.

One of the basic middleware services of sensor networks is network-wide time synchronization. Time synchronization helps to keep the data consistent by resolving redundant detection of the same event, it supports coordination and

communication e.g. TDMA radio scheduling, and it supports common services in distributed systems such as cryptography schemes, database queries or distributed logging for debugging.

In this paper we introduce the Flooding Time Synchronization Protocol (FTSP) for WSN. Our goals are to achieve a network wide synchronization error in the micro-second range and scalability up to hundreds of nodes, while being robust to network topology changes and node failures. To achieve these goals the possible error sources were identified and systematically analyzed. The proposed algorithm compensates for the relevant error sources by utilizing concepts of MAC layer time-stamping [2], [16] and skew compensation with linear regression [1].While these ideas are not completely new, their unique combination and its effective implementation yield an order of magnitude better precision than existing approaches on the same platform. Furthermore, the utilized broadcast-based synchronization protocol along with the skew compensation scheme helps to keep communication overhead low. Finally, the implicit dynamic topology handling of the FTSP provides fast convergence and robustness.

The algorithm was implemented on Mica/Mica2 platforms running the TinyOS operating system [7]. A short overview of the target platform is given in Section 2. Existing time synchronization algorithms, with special emphasis on ideas utilized in FTSP, are surveyed in Section 3. The possible sources of error using radio message-based synchronization are described and analyzed in Section 4. We describe the proposed FTSP algorithm in details and perform an evaluation based on a large scale experiment in Section 5. In Section 6, we compare FTSP to existing algorithms. A different time synchronization method that works in networks with aggressive power management is presented in Section 7. Finally, an application using FTSP time sync is described in Section 8 and we offer our conclusions and plans for further improvements in Section 9.

2. The target platform

One of the most widely used WSN platforms is the Berkeley Mica2 mote [10], [7]. 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 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 [7], [3], an open source, event driven and modular OS designed to be used with networked sensors. TinyOS handles task scheduling, radio communication, 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 sync

Time synchronization algorithms providing a mechanism to translate between the local times of nodes or to synchronize the local clocks of the nodes in the network have been studied in the past. The most widely adapted protocol used in the internet domain is the Network Time Protocol (NTP) devised by Mills [4]. The NTP clients synchronize their clocks to the NTP time servers with the accuracy in the order of milliseconds. The time servers are synchronized by the external time sources typically using GPS. The NTP has been widely deployed and proved to be effective, secure and robust in 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 this protocol is suitable only for WSN applications with low precision demands.

The most important examples of existing time synchronization protocols developed for the wireless sensor network domain are the Reference Broadcast Synchronization (RBS) algorithm [1] and the Timing-sync Protocol for Sensor Networks (TPSN) [2].

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 yet to large multi-hop networks.

The TPSN algorithm first creates a hierarchical structure in the network and then performs the pair-wise synchronization along the edges of the structure. 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 [2]. The shortcoming of TPSN is that it does not estimate the clock drift of nodes, which limits its accuracy. Moreover, it does not support dynamic topology changes and its performance was experimentally verified in a small multi-hop network only.

4. Uncertainties in the 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 Schwabl [6] and later extended in [2].

(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 current processor load, the send time is highly 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. Tthis 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.

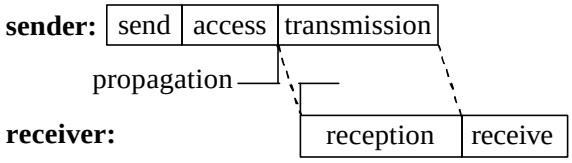

Figure 1 Decomposition of message delivery delay over a wireless link.

The RBS approach completely eliminates the send and access time, and with minimal OS modifications it is also possible to eliminate the receive time uncertainty. This leaves the mostly deterministic propagation and reception time in wireless networks as the sole source of error. The transmission time is not of concern in WSN as it overlaps with the reception time. The main strength of RBS is its broad applicability to commodity hardware and existing software in wireless networks.

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. The real novelty of TPSN however is that with a two-way handshake of synchronization messages the unknown propagation time can be calculated and compensated for on the fly.

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 the 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 sender to 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 antenna and radio chip converts it back to binary representation. Then the radio chip on the receiver side signals the microcontroller that a new piece is ready and can be read through some protocol. The timing of the actual data transfer mechanism between the microcontroller and radio chip is unimportant because data transfer can be done at any time before a new piece arrives. 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 one microsecond (waiting for the microcontroller to finish the currently executed instruction), however when interrupts are disabled this delay can be 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 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 [2].

(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.

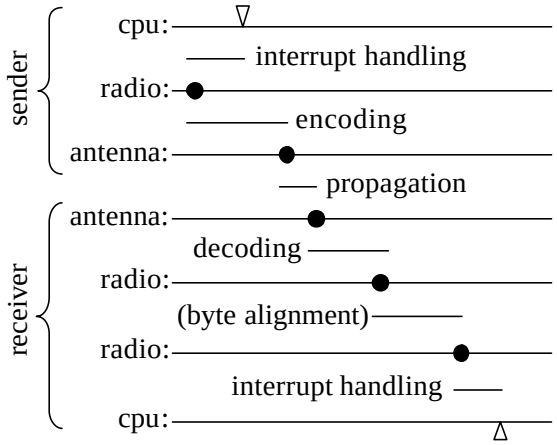

Figure 2 The timing of the transmission of an idealized point in the software (cpu), hardware (radio chip) and physical (antenna) layers of the sender and receiver.

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 scale 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 microcontroller 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.

On the Mica2 platform, the interrupt handling time is around 5µs depending on the length of the code path between the start of interrupt hander and the part that records the local time. However, with less than 2% probability, the interrupt handling time can be as high as 30µs. The sum of encoding and decoding times are 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.

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.

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. The error of the synchronization is in the micro-second range, and the protocol is scalable for hundreds of nodes, while being robust to network topology changes and node failures.

The FTSP achieves time synchronization between a sender and possibly multiple receivers utilizing a single radio message time-stamped at the both the sender and the receiver sides. The detailed analysis of the error sources suggests MAC layer time-stamping, as observed by [culler], [2]. However, a precise time-synchronization at a single point in time is a partial solution only. Compensation for the clock drift of the nodes is inevitable to achieve high precision and low communication overhead. The FTSP estimates the clock drift using linear regression (as suggested in [1]).

Typically, WSNs operate in areas larger than the broadcast range, therefore the FTSP needs to support a multi-hop synchronization. A single, dynamically (re)elected node, called the root of the network, maintains the global time and all other nodes synchronize their clocks to the local clock 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 the explicit hierarchical structure proposed in [2]. This saves the extra communication necessary to establish the hierarchical structure and is more robust against node failures and other topology changes.

5.1 The FTSP time-stamping

The FTS Protocol synchronizes the receiver to the time provided by the sender of the radio message. A radio broadcast is used to allow synchronization of multiple receivers using just one radio message. The broadcasted message contains the sender time-stamp which is the global time when sending a certain byte. The receivers get the corresponding local time from the local clock when receiving the message. This way one broadcast message provides a synchronization point (global, local time pair) to each of the receivers. Unlike the RBS and TPSN protocols, 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.

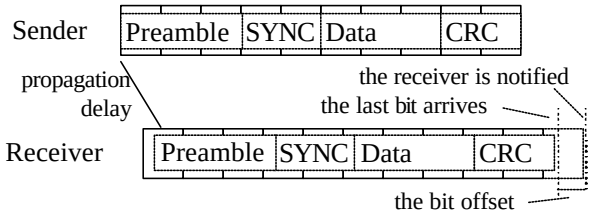

Figure 3 Transmitted packets over the radio channel. Full lines represent the bytes of the buffer and the dashed lines are the bytes of packets.

Wireless message transmission starts with the transmission of preamble bytes, followed by SYNC bytes, then with a message descriptor followed by the actual message data, and finally ends with CRC bytes. During the transmission of the preamble bytes the receiver radio chip 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 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 send and receiver side. 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, which can be calculated from the transfer rate. The jitter of interrupt handling time is mainly due to program sections disabling interrupts on the microcontroller for short amount of times. This error is not Gaussian and successfully can be eliminated by taking the minimum of the normalized time stamps. The jitter of encoding and decoding time can be combated by taking the average of these 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. The number of bytes put an upper limit on the achievable error correction using this technique. However, only with 6 time stamps, the time stamping precision can be improved from 25µs to 1.4µs on the Mica2 platform.

5.2 Dealing with clock drifts, one hop scenario

A single time-stamp would be sufficient to synchronize two nodes if the offset of their local times were constant. However, the frequency differences of the crystals used in Mica2 motes introduce drifts up to 40µs in one second time period. This would mandate continuous re-synchronization with a period of one second to keep the error in a micro-second range, which is not feasible in our domain. Therefore, we need to estimate the drift of the receiver clock with respect to the sender clock.

If the short term stability of the clocks is good, the offset between the two clocks changes in a linear fashion. We verified the stability of the 7.37 MHz Mica2 clock by periodically sending a reference broadcast, which was received by two different motes. The two motes time-stamped the message 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 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 on-line; it is used here as a reference to evaluate on-line solutions.

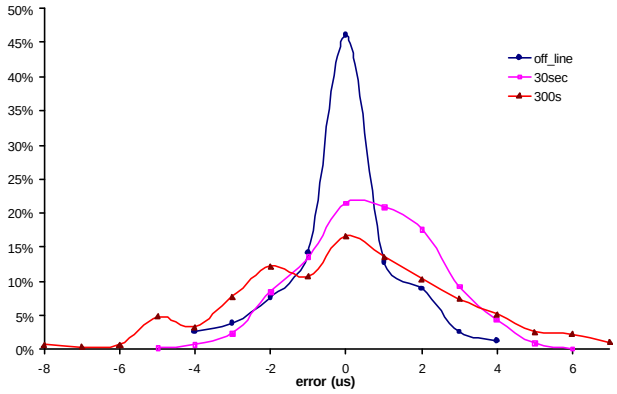

Figure 4 The distribution of the errors of linear-regression (LR): o*ff_line* refers to off-line LR, *30sec* refers to time sync interval 30s, query interval 23s, and *300s* refers to time sync interval 300s, query interval 93s.

The on-line linear regression needs to identify the trend of the global time relative to the local time from the data points received in the past. Moreover, as a consequence of the memory constraints of the platform, only a limited number of data points can be stored. 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*. *B* estimates the drift of its local clock using linear regression on the past 8 data

points; and both *A* and *B* respond to reference broadcasts with period *t* by time-stamping them with their estimate of the global time, and sending these time-stamps to a base station. 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.

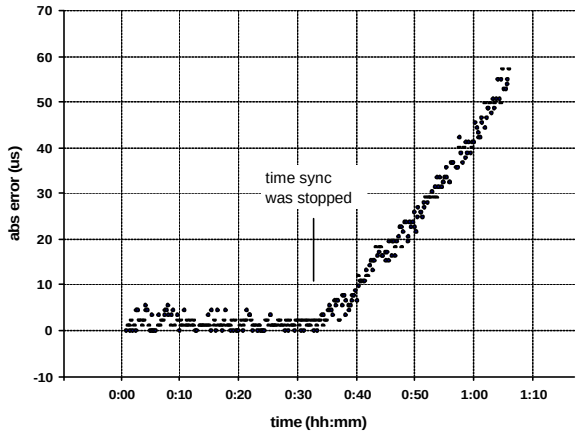

Figure 5 Time synchronization error between two motes. The time synchronization was stopped after 30 minutes. The initial small error of the skew estimate results in increasing error over time.

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 of the accuracy requirements, can go up to several minutes.

5.3 The multi-hop in FTSP

Most elaborate WSN applications use networks larger than one hop in radius, thus multi-hop synchronization is necessary to achieve network-wide time synchronization.

A possible solution to the problem is to provide a fraction of the motes with external synchronization methods, e.g. GPS sensors in such a way that all other motes are one hop away from them. However, this solution is cost prohibitive for most systems. The proposed multi-hop FTSP can synchronize the network without external time sources, provided that each node has a unique identifier, the node ID.

The global time in the multi-hop FTS Protocol is driven by the local clock of a single node, called the *root*. The global time is diffused into the network by each node periodically broadcasting its own global time estimate. Using a modified version of the one-hop synchronization scheme described in Sections 5.1 and 5.2, motes continuously synchronize themselves to (possibly multiple) motes that are closer to the root than themselves. The protocol defines how to handle redundant information from different sources, how to elect a root, and provides a mechanism to overtake the responsibility of the root by another node if the root fails.

- *The election problem:* Since there is no dedicated node in the network to provide time reference information, the root must be elected each time the network is started. The election process utilizes the unique IDs of the nodes. When a node does not receive time sync messages for a period of time, it declares itself to be the root and eventually starts sending time sync messages. It is possible, of course, that more than one node declares itself the root of the network. The FTSP resolves this problem by electing the mote with *the lowest ID* as the root of the network in the following way. All motes remember the ID of the root, to which they are currently synchronized to, in a local variable, called *myRootID*. If a node is root, then this variable holds its own node ID. Time synchronization messages contain a field, called *rootID*, which stores the *myRootID* of the sender. The time synchronization message is discarded by the receiver if the *rootID* in the message is higher than the *myRootID* of the receiver. On the other hand, if the *rootID* is smaller than the *myRootID*, then the *myRootID* of the receiver is set to *rootID*. In this case, if the receiver declared itself the root node of the network previously, it becomes a regular node at this point.

This guarantees that eventually no more than one root remains in the network.

- *Handling redundant information:* The global time information can arrive to a node from the root along different routes, and the precision of the global time estimate may deteriorate over time as it is passed along the network. Moreover, the linear regression of a limited number of data points more accurately estimates the offset and skew of the local clock if the data points are distributed over a longer period of time. Therefore, each node has to select an appropriate subset of the received time synchronization messages that are entered into the eight-element regression table, and used for calculating the regression line. The FTSP employs sequence numbers for this purpose. Each time synchronization message contains a *seqNum* field, which is set and incremented by the root each time it sends a new message. Other nodes maintain a *highestSeqNum* local variable, which contains the highest sequence number of those received messages whose *rootID* is *myRootID*. These nodes set the *seqNum* field of their broadcasted messages to the current value of their *highestSeqNum*. Consequently, a node considers a time sync message new if the *rootID* of the message is less than or equal to *myRootID* and the *seqNum* is greater than *highestSeqNum*. '*New*' time synchronization messages are entered into the regression table, others are discarded. This protocol guarantees that only one data point will be entered into the table for each *rootID* and *seqNum* pair, namely the one that arrived first. Since the first synchronization message probably took a short and good quality path (although it is *not* enforced that it has the smallest possible hop-count), it is likely more accurate than the following messages having the same *seqNum*.

- *The node and link failure:* Errors caused by failing hardware or drained batteries are the norm rather than the exception in WSN and the FTSP needs to be robust against these failures. Periodic broadcasting of time synchronization messages handles the regular node and link failures well, but does not help when the root fails. The following mechanism, similar to the initial leader election process, is used to replace the root in case of its failure. Each node remembers the most recent time when the root was active. A good approximation of this is the time when the last new time sync message arrived (*highestSeqNum* was changed). Each node will time out if the root has not been active for a certain time period and will declare itself to be the root. Therefore, all nodes in the network will eventually time-out, and the election process will resolve multiple root conflicts. It is clear that inconsistent timing information would be produced during this election process since multiple sources of the global time would exist. To avoid the inconsistency, nodes keep their old global time estimates and the new root sends its global time estimate instead of its local time as a new global time. The new global time is very close to the old global time this way and the network does not get out of synchronization during the root reelection.

- *Topology changes:* Nodes join and leave the network dynamically, and some of them are possibly mobile. The only assumption we make here is that the network remains connected at all times. The effect of removing the root from the network was explored before. Another problematic case is when a new node *M* with smaller ID than the root is switched on. If *M* transmitted its local time as a new global time immediately after switching on, all the nodes in the network would get out of the synchronization. Therefore, each newly introduced node first waits for a certain time period, gathers data for the linear regression and determines the offset and skew of its own local clock from the global time. This way *M* is able to overtake the role of the old root and send a global time that is close to the old global time in such a way that the network does not get out of synchronization. Even if *M* has a higher node ID than that of the current root, it still waits for a certain time period before it rebroadcasts time sync messages to avoid sending erroneous global time.

Since time sync messages with the lowest *rootID* and highest *seqNum* flood the network, topology changes do not hinder the algorithm provided the network stays connected.

- *The convergence of the algorithm:* The speed of information propagation (root ID and global time) to all nodes of the network is very important in the case of node failures, system startup and resume from powered down mode (see Section 7). Node failures can be handled very smoothly, as described

in the previous section, because the remaining nodes are already synchronized. The initial phases of switching on or waking up the system from sleep mode are more critical. There exists a physical limit on the time it takes for the network to synchronize. If *period* is the time period every node broadcasts a time sync message (note that the individual motes are transmitting asynchronously), and *radius* is the maximum hop-count of nodes to the root, then the expected value of time it takes the network to learn about the identity of the root is *radius*period/2*. To get an estimate of both the skew and offset of the local clock, nodes need at least two data points in the regression table. Therefore, it takes approximately *radius*period* time to synchronize all the nodes in the network. This illustrates a tradeoff between power consumption and speed of convergence: decreasing the *period* increases the number of messages sent in a certain time period but allows faster convergence.

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 [8]). 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.

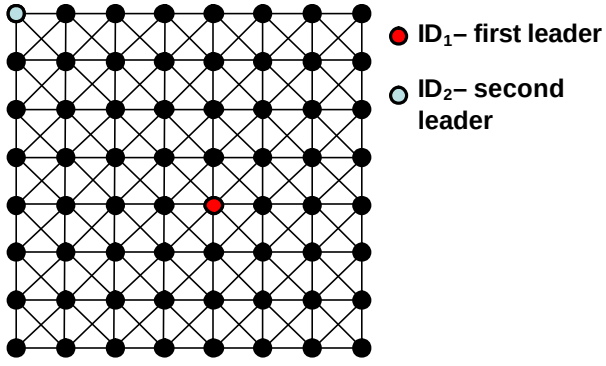

Figure 6 The layout and links of the experimental setup: 64 motes are distributed in 8 rows and 8 columns and each node can only communicate with its direct neighbors. The maximum hop distance from the first and the second leader are 4 and 7 respectively.

The experiment scenario involves 64 Mica2 motes deployed in 8x8 grid in such way that each mote can communicate only with its direct neighbors. 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 6. This means that ID_1 will eventually become the root of the network and ID_2 will become the root if ID_1 dies. The maximum hop distance between ID_1 and ID_2 represents the worst case scenario if the root ID_1 dies.

Two other motes were used in the experiment, the reference broadcaster, and the base station. Their function was the same as described in Section 5.2. The topology of the 64 nodes network was enforced in software and so all the nodes could be placed within the radio range from the reference broadcaster. This way the base station and the broadcaster could talk directly to all 64 nodes and no multi-hop routing was necessary.

Each of 64 nodes broadcasted one time synchronization message per 30 seconds. The reference broadcaster queried the global time from all nodes in the network once per 30 seconds and the base station collected the responses.

We performed the following experiment:

- at 0:00 all motes were turned on;
- at 0:41 the root with ID_1 was switched off;
- from 1:12 until 1:42 randomly selected motes were switched off and back on, one per 30s;
- at 1:47 the motes with odd node IDs were switched off (half of the nodes are removed);
- at 2:02 the motes with odd node IDs were switched back on (100% new nodes are introduced);
- at 2:13 the second root with ID_2 is switched off;

Even though there were 64 nodes in the network, at any time typically only 80.0% of them succeeded to reply to the reference broadcaster due to radio collisions. The nodes reported back to the base station whether they were synchronized (i.e. have 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. We analyzed the time synchronization error by first calculating the average *G* of reported global times and then for each node calculating the difference between the reported global time and *G*.

Consequently, we computed the average and maximum of the absolute values of these differences, called the *average* and *maximum time synchronization error*, respectively. The resulting graph is shown in Figure 7.

Even though there were 64 nodes in the network, at any time typically only 80.0% of them succeeded to reply to the reference broadcaster due to the radio collisions. The nodes reported back to the base station whether they were synchronized (i.e. have 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. We analyzed the time synchronization error by first calculating the average *G* of reported global times, and then for each node calculating the difference between the reported global time and *G*. Consequently, we computed the average and maximum of the absolute values of these differences, called the *average* and *maximum time synchronization error*, respectively. The resulting graph is shown in Figure 7.

The beginning of the experiment has shown the convergence of the algorithm: during the first 3 minutes the nodes were not synchronized, because none of them declared itself to be the root. The nodes were switched on approximately at the same time, so in the next few minutes many of them timed out and became the roots of the network. This was the reason why the average and maximum synchronization errors soared during this time period. However, after the 6th minute the election process has completed and only a single root remained (ID_1).The number of synchronized nodes started to grow steadily, and the average and maximum errors became approximately 2.5µs and 7.5µs, respectively. Complete synchronization has been achieved in 10 minutes as indicated by the percentage of synchronized motes reaching 100%.

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 slight deterioration of the maximum and average errors until all nodes calculated more accurate drift estimates based on the messages broadcasted by the new root. In the last part of the experiment some of the nodes were removed and new ones were introduced. 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. Worth noticing is also the fact that the network recovered faster after the root ID_2 was switched off than after ID_1. This was also expected since the root which took over after ID_2 was 1 hop away from ID_2.

The 64-mote 7-hop network synchronized in 10 minutes and the average time synchronization error stayed below 11.7µs. If we divide it by number of hops, we get the average error of 1.7µs per hop. The maximum time synchronization error was below 38µs, 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.

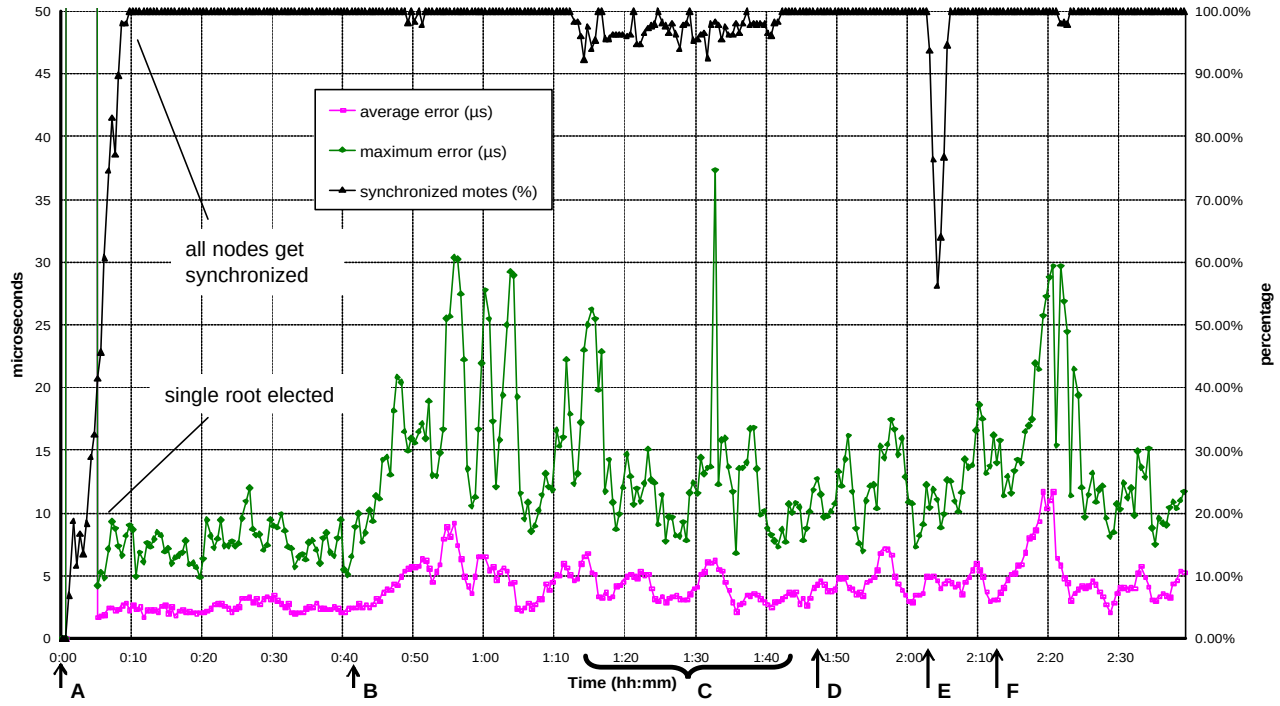

Figure 7 An 8x8 grid experiment shows the percentage of synchronized nodes, the maximum and average error (the maximum and average absolute offset from the average reported global time). The nodes were switched on at time A, the root ID_1 was switched off at B, multiple motes were randomly switched off and back on during the C, half of the motes were switched off at D, the same motes were switched back on at E and finally the root ID_2 was switched off at F.

The RBS approach time-stamps messages only on the receiver side; therefore, it eliminates the access and the send times. The published method in [1] does not compensate for byte alignment, but that could be easily incorporated. The main achievement of the RBS time-stamping of a reference broadcast by two receivers 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.

The TPSN approach 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 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.

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. For multi-hop case the average error was 11.7µs in a 7-hop network, resulting in a 1.7µs 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 [2]. FTSP also utilizes less network

resources either both 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). The robustness of the protocol was demonstrated by the harsh experiment described in Section 5.5. Unfortunately, no similar data is readily available for TPSN or RBS for comparison.

7. Time synchronization in powered down sensor networks

The FTSP described above makes use of continuous time synchronization where every mote periodically broadcasts time synchronization messages. Although measurements suggest that the broadcast period can be as high as several minutes, depending on the accuracy requirements (see Figure 4), the protocol still requires continuous operation of the motes, thus limiting the lifetime of the application. The approach we previously described for both the single and multi-hop cases is definitely not applicable for systems operating over several weeks or months. These applications are not required to be continuously active, and are powered down most of the time to save energy. The question arising naturally is whether continuous time synchronization is really necessary.

As pointed out in [1], in many cases post facto synchronization is enough, no continuous synchronization is required. Especially those systems collecting data or reacting to rare events, but requiring exact time measurements belong to this case.

A possible way of post facto synchronization is described in [1], utilizing explicit pair-wise synchronization after message passing. We propose an alternative method embedded into the routing protocol which does not require any additional message exchange apart from the routing messages.

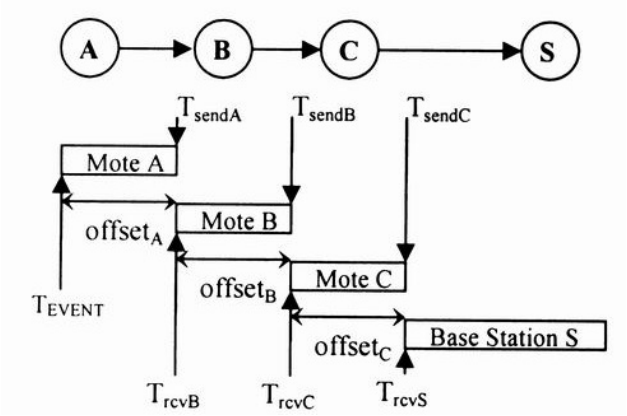

Figure 8 Estimation of detection time T_{EVENT} can be iteratively determined along a routing path *A,B,C,S* as $T_{rcvS} - offset_A - offset_B - offset_C$

The basic problem is the following: a sensor detects an event and the target node needs to know the time of the event in its own local time. The sensor and the target nodes may be several hops apart from each other. Still, it is possible to solve the problem without any explicit time synchronization in the network. An implicit synchronization may be performed during the routing process.

Along with the sensor reading a radio message includes an *age* field, which contains the elapsed time since the occurrence of the event. This additional information adds only a very small overhead to the message. Each intermediate mote measures the *offset*, which is the elapsed time from the reception of a sensor reading till its retransmission. The *age* field is updated upon transmission using the time stamping method described in Section 5.2. When the sensor reading arrives to the destination, the *age* field contains the sum of the offsets measured by each of the motes along the path. The destination node can determine the time of the event by subtracting *age* from the time of arrival of the message. The concept is illustrated in Figure 8: An event is detected at node *A* at time instant T_{EVENT}, then a notification message is sent to destination node *S* through nodes *B* and *C*. The message delays at the nodes are $offset_A$, $offset_B$, and $offset_C$, respectively. The message arrives to *S* at time instant T_{rcvS}, containing an *age field* $offset_A+offset_B+offset_C$. The time of the event can be calculated as $T_{EVENT} = T_{rcvS} - age$.

One possible problem with this approach is that the time measurement units of the intermediate nodes are not of the same length, because of the slight differences in their clock frequencies. Since this method does not compensate for skew errors, significant error can accumulate if the routing of the sensor reading takes a long time. According to MICA2 platform specification, the clock skew error is less than 40µs per second. Thus the worst-case post-facto synchronization error can be estimated as $4*10^{-5}T_R$, where T_R is the worst-case time of the message routing.

This time synchronization algorithm can be further refined by exploiting the usual properties of certain wireless routing protocols. Because of unreliable radio channels the same radio message may be rebroadcasted several times at intermediate nodes, and it can arrive to the base station multiple times along different paths. Even though these multiple messages hold the same sensor reading, the attached elapsed time can vary, mainly caused by the different clock frequencies of the nodes along the different routes. The destination node can use a statistical analysis of the received elapsed times to get a better estimate of the time the event occurred.

The main advantage of the proposed integrated time synchronization and routing algorithm is that it does not require additional radio messages, and the overhead imposed on the original routing messages is very low.

8. Applications

The FTSP algorithm was excessively tested as a component of a countersniper application. 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 [15]. 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 MICA2 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.

9. Conclusions and further improvements

We have introduced 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 1.7µs per hop in the multi-hop case were shown by providing experimental results. This performance is significantly better than those of other existing time synchronization approaches on the same platform.

Furthermore, the protocol 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.

Further work may focus on more precise time estimates by time stamping multiple bytes of the message during transmission. Taking an average of these and working with fractional time-stamps can reduce the time resolution error.

Another research area is to improve the convergence of the multihop 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.

10. References

[1] J. Elson, L. Girod and D. Estrin, "Fine-Grained Network Time Synchronization using Reference Broadcasts," P*roceedings of the fifth symposium OSDI '02*, December 2002.

[2] S. Ganeriwal, R. Kumar, M. B. Srivastava, "Timing-Sync Protocol for Sensor Networks," *SenSys '03*, November 2003

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

[4] D. L. Mills. "Internet Time Synchronization: The Network Time Protocol" In Z. Yang and T. A. Marsland, *Global States and Time Distributed Systems*. IEEE Computer Society Press, 1994

[5] J. E. Elson, "Time Synchronization in Wireless Sensor Networks", dissertation, University of California, Los Angeles 2003

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

[7] J. Hill and D. Culler, "Mica: A Wireless Platform for Deeply Embedded Networks", IEEE Micro., vol 22(6), Nov/Dec 2002, pp 12-24.

[8] The TinyOS implementation of the FTSP: http://cvs.sourceforge.net/viewcvs.py/tinyos/minitasks/02/vu/tos/lib/TimeSync/

[9] J. M. Kahn, R. H. Katz and K. S. J. Pister, "Mobile Networking for Smart Dust", ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99), Seattle, WA, August 17-19, 1999

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

[11] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson , "Wireless Sensor Networks for Habitat Monitoring", ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, Georgia, September 2002

[12] L. Schwiebert, S. K.S. Gupta and J. Weinmann, "Research Challenges in Wireless Networks of Biomedical Sensors", SIGMOBILE 2001

[13] M. Srivastava, R. Muntz and M. Potkonjak, "Smart Kindergarten: Sensor-based Wireless Networks for Smart Developmental Problem-solving Environments", SIGMOBILE 2001

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

[15] Ledeczi, A. et al.: "Sensor Network-Based Countersniper System," to appear in the Proc. of Mobisys, Boston, MA, June 2004

[16] Woo, A. and Culler, D.: "A Transmission Control Scheme for Media Access in Sensor Networks," Proc. Mobicom, 2001