Robust Multi-Hop Time Synchronization in Sensor Networks

Miklos Maroti, Branislav Kusy, Gyula Simon and Akos Ledeczi
The Institute for Software Integrated Systems, Vanderbilt University
Nashville, TN, U.S.A.

Abstract *– The possibility of establishing the chronology of events in a widely distributed network, or even stronger, the clock synchronization of all nodes in the network is often needed for applications of wireless sensor networks (WSN). In this paper we describe the Flooding Time Synchronization Protocol (FTSP) that provides clock synchronization service in such networks. The protocol was designed to utilize low communication bandwidth, scale well for medium sized multi-hop networks, and be robust against topology changes and node failures. The robustness of the protocol is due to the periodic radio broadcast of synchronization messages and the implicit dynamic topology update. MAC-layer time-stamping, comprehensive compensation of errors and linear regression are used to achieve high accuracy of the clock synchronization. The data from a comprehensive multi-hop experiment shows that the average network-wide synchronization error is less than two microseconds per hop. The protocol was further validated as part of our countersniper system that was field tested in a US military facility.*

1. Introduction

The unique characteristics of the emerging wireless sensor network (WSN) domain, especially the severe resource constraints associated with the individual nodes, demand the reevaluation of distributed 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 also supports more effective coordination and communication. Emerging applications of WSN that use time synchronization are described in [8], [9], and [10]. However, the memory, bandwidth, computational and power constraints of the individual nodes, together with the requirements for robustness to node failures prohibit the use of traditional algorithms such as Mill's NTP [4].

There are also numerous time synchronization protocols developed specifically to WSN applications with the special requirements of sensor networks in mind. The best know protocols are the RBS [1] and the TPSN algorithms [2].

The RBS approach uses broadcasted synchronization messages to synchronize the receivers to each other (but not to the sender). Therefore RBS needs time-stamping only on the receiver side, thus, it eliminates error sources as the access and the send times. RBS does not compensate for byte alignment, an important error source on the targeted MICA platform. The greatest innovation of the RBS time-stamping of a reference broadcast by two receivers is that it eliminates all sender side non-determinism.

The TPSN approach eliminates errors caused by access time, byte alignment time and propagation time by making use of the implicit acknowledgments. Additional accuracy is gained by time-stamping each radio message multiple times and averaging these time-stamps. Currently the Mica2 platform does not support implicit acknowledgments making TPSN hard to port to the platform. Another disadvantage of TPSN is that a message-acknowledgement pair is used for a pair of nodes prohibiting the use of message broadcasts resulting in higher communication overhead.

The reported accuracy of the RBS time-stamping is ~11μs. Least square linear regression is used to compensate the effects of clock drifts which

results in 7.4μs average error between two motes after one minute. The multi-hop scenario involves the local time transferring through the intermediary nodes. However, time synchronization was carried out PDAs, not the motes. The authors of TPSN algorithm implemented both TPSN and RBS on the Mica platform using a 4 MHz clock for time-stamping. The resulting pair wise average errors were 16.9μs for TPSN and 29.1μs for RBS.

The goal of the propose Flooding Time Synchronization Protocol (FTSP) is to achieve a network wide synchronization error in the micro-second range and to provide scalability up to hundreds of nodes, and at the same time being robust to network topology changes and node failures. The algorithm utilizes broadcasting to obtain time synchronization reference points between the sender and its neighbors. Nodes broadcast time synchronization messages periodically, and synchronize their clocks to that of an elected leader. When the leader fails sending messages for a certain time period, a new leader is elected automatically.

To achieve the low synchronization errors we use the following concepts to compensate for the errors: MAC layer time-stamping [7], [2] with several jitter reducing techniques, and skew compensation with multiple time-stamps and linear regression [1]. While these ideas are not completely new, their unique combination and its effective implementation yield significantly better precision than the existing algorithms RBS [1] and TPSN [2] (implemented on the Mica/Mica2 platforms [5] running the TinyOS operating system [3]). Furthermore, the utilized broadcast-based synchronization protocol along with the skew compensation scheme help keep communication overhead low. Finally, the implicit dynamic topology handling of the FTSP provides fast convergence and robustness.

2. Flooding Time Synchronization Protocol

The FTSP uses a single radio message time-stamped at both the sender and the receiver sides to achieve time synchronization between a sender and all of its neighbors at once. The detailed analysis of the error sources suggests MAC layer time-stamping, as observed by [7], [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 [1].

In real-world applications typically multi-hop synchronization is required. In FTSP 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. There is no explicit hierarchical communication structure as proposed in [2], but instead the nodes form a dynamically changing ad-hoc structure to transfer the global time from the root to all the nodes. This solution provides more efficient use of the communication channel and it also is very robust against node failures and other topology changes.

The basic element of the FTSP is the accurate time stamping. The FTS Protocol synchronizes the receiver to the time provided by the sender of the radio message. The radio message contains the sender time-stamp which is the global time when sending a certain byte. The receivers record the local time of receiving the corresponding byte. Using a radio broadcast, one radio message provides a synchronization point (global, local time pair) to multiple receivers. Unlike the RBS and TPSN protocols, the time stamp of the sender must be embedded in the currently transmitted message.

Wireless message transmission starts with the transmission of preamble bytes, followed by SYNC bytes, message descriptor bytes, the actual message data, and finally CRC bytes. During the transmission of the preamble bytes the receiving radio chip synchronizes itself to the carrier frequency of the incoming signal. From the SYNC bytes the receiver can calculate the bit offset from the sender. This is needed 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 information about 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 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 the boundaries of bytes following the SYNC bytes, by both the transmitter and the receiver.
The jitter of interrupt handling time is mainly due to program sections disabling interrupts on the microcontroller for short amount of time. This error is not Gaussian and can be successfully eliminated by taking the minimum of the time stamps. The jitter of encoding and decoding times can be alleviated by taking the average of these 'interrupt error' corrected 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 we timestamp puts 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 25μs to 1.4μs on the Mica2 platform.

A single synchronization message 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 and the linear regression can be used to estimate the offset. We verified the stability of the 7.37 MHz Mica2 clock by sending a reference broadcast, which was received by two different motes. Both motes reported the local times of arrival of the broadcast. We calculated the offsets of the reported times and determined the best fit line for these offsets. The average error of the linear regression was 0.95μs proving the short-term stability of clock crystals.

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. We implemented the on-line regression on Mica2 and changed the previous scenario to send the global times instead of the local times. The measured average absolute error of the global time prediction was 1.48μs in an 18-hour experiment.

From the long-term point of view, the relative drift of two clock crystals can change when the temperature, humidity, or battery power level change. The required resynchronization interval to reach the desired precision is an important design parameter. Our experiments showed that this resynchronization period can go up to several minutes.

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 earlier; motes continuously synchronize themselves to the motes that are closer to the root than themselves. The protocol defines how to handle 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.

Since there is no node in the network dedicated 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. If a node is root, then this variable holds its own node ID. Time synchronization messages contain the root ID of the sender. The time synchronization message is discarded by the receiver if the root ID in the message is higher than the known root ID of the receiver. On the other hand, if the received root ID is smaller than the currently known root ID, then the root ID of the receiver is reset to the just received root ID, and the receiver becomes a regular node at this point, if it was a root before. This guarantees that eventually no more than one root remains in the network.

The global time information from the root can arrive to a node along different routes. Moreover, the precision of the global time estimate may deteriorate over time as it is passed along the network. Therefore, each node has to select an appropriate subset of the received time synchronization messages that are entered into its regression table. Each time synchronization message contains a sequence number, which is set and incremented by the root each time it sends a new message. Other nodes remember the highest sequence number received from the root so far. These nodes broadcast messages containing this number. Consequently, a node considers a time sync message new if its sequence number is greater than the highest sequence number of the node. ‘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 root ID and sequence number pair, namely the one that arrives first.

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 (the stored highest sequence number 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.

Nodes may 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.

Note that since time sync messages with the lowest root ID and highest sequence number flood the network, topology changes do not hinder the algorithm provided the network stays connected.

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. Node failures can be handled very smoothly 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 T_s is the time period every node broadcasts a time synchronization message 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 $radius*T_s/2$. 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 $radius*T_s*2$ time to synchronize all the nodes in the network. This illustrates a tradeoff between power consumption and speed of convergence: decreasing the synchronization period increases the number of messages sent in a certain time period, but allows faster convergence.

3. Evaluation

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 [6]). 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.

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. 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. The nodes time-stamped the broadcaster's messages with the global time and sent these global time-stamps to the base station. The base station only forwarded all the data to the laptop. 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.

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.

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;

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 1.

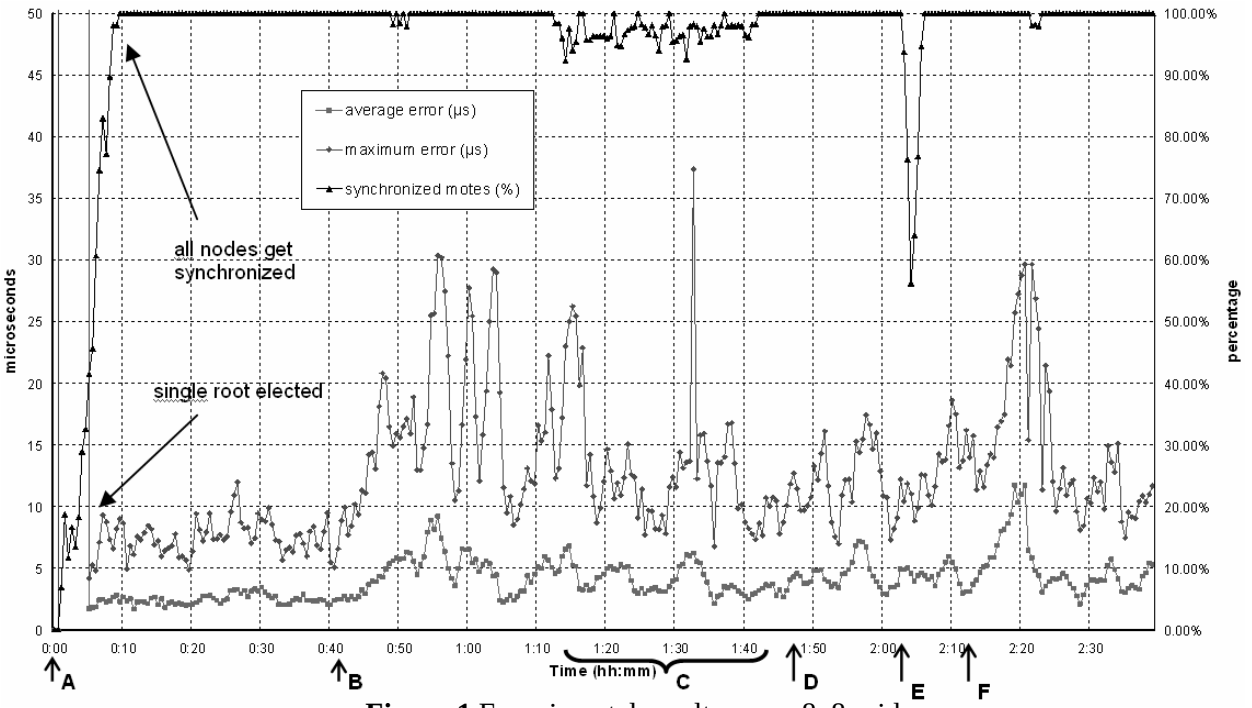

Figure 1 Experimental results on an 8x8 grid.

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**.**

4. Conclusions

Wireless sensor network applications (WSN), similarly to other distributed systems, often require a scalable time synchronization service enabling data consistency and coordination. We have presented our time synchronization approach for WSN, the Flooding Time Synchronization Protocol.

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 both an experimental setup and a real-world application [11]. Although our protocol was implemented and tested for a specific platform, the approach used is general and we believe it can be applied to other WSN platforms and different operating systems.

5. 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. Hill and D. Culler, "Mica: A Wireless Platform for Deeply Embedded Networks", IEEE Micro., vol 22(6), Nov/Dec 2002

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

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

[8] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, J. Anderson , "Wireless Sensor Networks for Habitat Monitoring", WSNA'02, Atlanta, GA

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

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

[11] Vanderbilt Shooter Localization website: http://www.isis.vanderbilt.edu/projects/nest/applications.html