Time Synchronization for ZigBee Networks

Dennis Cox, Emil Jovanov, Aleksandar Milenkovic
Electrical and Computer Engineering Department
University of Alabama in Huntsville
Huntsville, AL 35899 USA
jovanov@ece.uah.edu

Key Words: Wireless Sensor Networks, Synchronization, ZigBee.

***Abstract*—Time synchronization is essential for most network applications. It is particularly important in a Wireless Sensor Network (WSN) as a means to correlate diverse measurements from a set of distributed sensor elements and synchronize clocks for shared channel communication protocols. Wireless sensors are typically designed with very stringent constraints for size, cost, and especially power consumption. The Flooding Time Synchronization Protocol (FTSP) was developed explicitly for time synchronization of mesh-connected wireless sensor networks. However, ZigBee can also accommodate master-slave networks that can be more power-efficient. We optimized the FTSP for master-slave WSNs and implemented it using TinyOS 1.1.8 and ZigBee-compliant hardware. Our approach allows better synchronization and reduced power consumption of wireless nodes. In this paper we present implementation and experimental results.**

I. INTRODUCTION

Computing technologies have consistently followed a general trend of becoming more and more distributed and deeply embedded into the environment. In keeping with this trend, the dream of truly ubiquitous computing will be empowered in part by a vast array of tiny computing elements wirelessly connected together providing detailed information about the world around us and acting upon the environment as well.

This is the role of a Wireless Sensor Network (WSN) as a collection of miniature computers designed for an extremely resource-constrained environment. They must satisfy the following characteristics:

- small size in order to be unobtrusive or hidden and not clutter up the landscape
- very cheap so they can be affordably deployed in large numbers
- extremely power efficient. Power efficiency extends battery life or allows energy scavenging from the environment. Typical examples include solar energy, energy of vibrations, etc.

ZigBee is an emerging standard for wireless communication target for use in WSNs. It was specifically designed with the severely resource-constrained sensor in mind. Together with the IEEE 802.15.4 [4] standard which defines the physical and MAC-layer interface, ZigBee has very low power consumption requirements. 802.15.4 networks can operate in either master-slave or peer-to-peer configuration. The master-slave would be common in many sensor-based applications where a number of very limited slaves are controlled by a master controller, such as home automation or security systems. 802.15.4 provides a number of features to help conserve power, such as beacons, long superframe cycles, and guaranteed time slots with power efficient idle modes on sensors.

ZigBee supports both mesh network and a star network topology in which a node acts as master for a number of other slave nodes. The master sends periodic beacon messages to the slaves, providing regular windows of time for the nodes to communicate. This mode allows the nodes to sleep during inactive time slots, which is ideal for certain networks where it is critical to preserve power.

This paper is organized as follows. We present common time synchronization mechanisms for wireless sensor networks in the second section. Our implementation of time synchronization for Telos platform in ZigBee environment is given in the third section; experimental results and conclusions are presented in the last section.

II. TIME SYNCHRONIZATION

Time synchronization of distributed computing elements is a common requirement for many distributed applications. Precise time synchronization can be essential in a WSN to facilitate group operations, such as sensor localization, data aggregation, distributed sampling, source localization, etc. Synchronized time stamps can be critical for proper correlation of sensor information from the various sources. In addition, synchronized clocks are essential for shared channel communication protocols, such as Time Division Multiplexing.

Each application that relies on time synchronization has a different set of requirements. Based on these requirements, a potential synchronization mechanism should be evaluated

using several metrics [2] by answering questions such as these: How much precision is needed? Do the network nodes need to remain synchronized all the time or can they just achieve synchronization when needed (as when several nodes need to compare the detection time of a single event)? Do the nodes occupy a large geographic area, and does that area need to be completely covered? How much time and power are available for the node to expend towards synchronization? What is the cost to implement synchronization and does it require any special equipment or infrastructure? In a WSN composed of a vast number of tiny, battery-powered nodes, the two most important factors would probably be power efficiency and cost.

A number of protocols and algorithms exist and have been implemented to provide time synchronization in computer networks. The Network Time Protocol (NTP) [7] is perhaps the most widely used. However it requires a significant amount of computation and computer resources, and it is not especially fault-tolerant, making it ill-suited for a WSN. Other protocols have been developed explicitly for WSNs such as the RBS [1] and TPSN algorithms [3].

We chose to implement the Flooding Time Synchronization Protocol (FTSP) that was developed at Vanderbilt University [5, 6]. This protocol was developed to demonstrate a means of providing network-wide time synchronization to a large network of wireless sensors. The FTSP was designed to accommodate network topology changes (which is necessary when the sensors are mobile) and to be robust despite the failure of individual nodes (a necessary consideration in a WSN). The two design features that seem especially valuable for a WSN are MAC layer time stamping for increased precision and skew compensation with linear regression to account for clock drift. The FTSP was demonstrated on a large network of 64 sensors using the Berkeley Mica2 mote.

The FTSP generates time synchronization with periodic time sync messages. The network can dynamically elect a root node. Whenever a node receives a time sync message, it rebroadcasts the message, thus flooding the network with time sync messages. The message itself contains a very precise timestamp of when the message was sent. The receiving node takes a timestamp when it gets the message. To remove sources of error, these timestamps are taken deep in the radio stack. Comparing with the timestamps from the last several messages received, the node computes a simple linear regression to allow it to account for the offset difference in its clock from global time as well as the relative difference in frequency (Fig. 1). This enables each node to maintain an accurate estimate of global time despite the fact that its local clock may be running slightly faster or slower than the global clock source.

The FTSP can provide a WSN with high-precision synchronization without requiring a lot of resource overhead. However, in some situations it could be optimized for greater

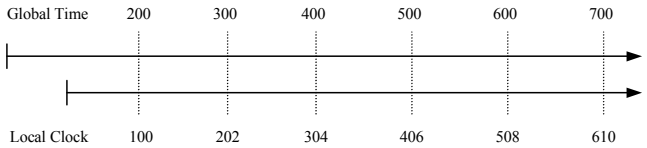

Figure 1. To synchronize to global time, a node must compute the offset between its local clock and the global time as well as the skew or rate at which its clock is drifting slower or faster than global time. The numbers shown are merely illustrative.

efficiency. A node in a WSN expends a significant amount of energy for wireless communications. The radio transmitter certainly consumes a lot of energy sending each message, but the receiver does too, even when it is just listening. Ideally, in a very energy-constrained network, a node will transmit as little as possible and even turn off its receiver until it expects to receive something. The ZigBee star network topology is well suited for this environment because of its master-slave hierarchy. Periodically, the master node transmits a beacon message to its slave nodes to maintain the communication link. Commonly the slave nodes may be tiny, simple sensors with a very limited power source. With this in mind, the FTSP could be optimized for a master-slave network such that only the master nodes transmit the periodic time sync messages and the slave nodes simply receive without having to transmit at all. In fact, a slave node could even disable its radio and enter low-power mode between messages. And highly accurate time synchronization could enable a slave node to sleep for a very long time and still wake up just before the next message was due, making the most of its limited energy supply.

III. IMPLEMENTATION OF TIME SYNCHRONIZATION ON TELOS PLATFORMS

In implementing the FTSP on the Telos platform, we assumed a master-slave configuration and a star network topology, taking advantage of the optimizations described above. Slight modifications to the program could allow a mote to operate in either a generic mesh network where time sync messages must be flooded or in a star network topology where only the master transmits the messages. It could even switch between the two modes of operation if network conditions warrant. Thus, the FTSP could be used in its original robust form or this simplified and optimized form as needed.

The Telos platform [9] is designed to serve as a node in a WSN. It comes equipped with a MSP430 microcontroller and a ZigBee-compliant ChipCon CC2420 intelligent wireless controller. It is also one of the platforms supported by the TinyOS operating system [10]. This is convenient because the FTSP was first implemented as a part of TinyOS [11]. However, porting the FTSP to Telos was not trivial, since the original implementation was based on the Mica2.

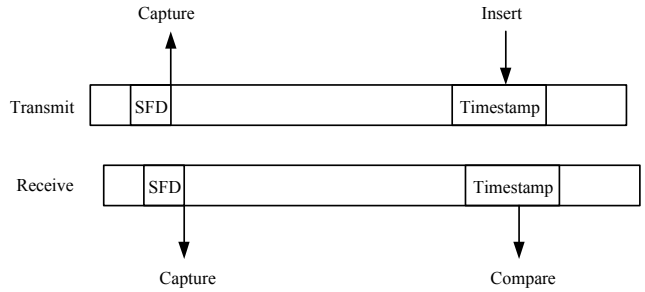

Figure 2. Illustrates a time synchronization message being transmitted and received. The transmitter captures a global timestamp at the end of the Start of Frame Delimiter (SFD) and inserts it into the message. The receiver captures a local timestamp at the end of the SFD and compares that with the timestamp embedded in the message. The messages are not perfectly aligned due to propagation delays.

For time synchronization to work, there must be a fixed point in time from which both sender and receiver can reference the timestamp in a given message. For a ZigBee message, this point is at the end of the Start of Frame Delimiter (SFD), as depicted in Fig. 2. The sender makes a timestamp immediately after it has transmitted the SFD and inserts the timestamp into the message (note that the message has already begun transmission when the timestamp is made and added to the message). The receiver makes a timestamp when it receives the SFD and stores it with the message. Later, the processor will compare the two timestamps to determine the offset between local time and global time and the degree to which the local clock is running at a faster or slower rate than the global clock. On the Telos platform, the wireless controller provides a signal to the microcontroller to indicate when the SFD byte has been received or transmitted. This signal was configured to generate a timer capture and an interrupt, enabling extremely accurate timestamps of each message. This is an improvement over even the Mica2 implementation, which creates very accurate timestamps deep in the radio stack.

Unlike Mica2 boards where the processor directly controls the wireless bit-stream, the Telos wireless controller provides FIFOs for transmit and receive data. The processor communicates with the controller using a synchronous peripheral interface (SPI). Generally, when transmitting a message, the processor loads up the transmit FIFO with the entire message and then enables transmission. However, the FTSP messages contain a timestamp that is generated after the message has begun transmission. To implement this on the Telos platform, most of the message is placed in the FIFO and transmission is enabled. When the SFD interrupt occurs, the captured timer value is retrieved and converted to a global timestamp. The timestamp is inserted into the message and the rest of the message is placed in the FIFO. Assuming this can all be done quickly enough, the entire message is transmitted properly. If however the process is too slow, the FIFO will underrun and the message transmission will abort. This is a real concern since ZigBee specifies a fairly speedy effective bit rate of 250 kbps.

Time measurements were taken to ensure adequate margin and instill confidence that the FTSP could run reliably without FIFO underrun. The SPI interface between the microcontroller and the wireless controller on the Telos platform runs at 500 kbps, twice as fast as the message is transmitted over the radio, so there seemed to be hope. It takes about 700 µs to transmit a time sync message and about 150 µs to calculate and insert the timestamp. It takes another 300 µs to send the second part of the message over the SPI and finish filling the FIFO. This means that the entire message makes it into the FIFO with about 250 µs to spare, which ought to be adequate.

The implementation for the Telos platform includes an interface that provides applications access to the time synchronization information. It can give the current global time, convert a local timestamp to global time, or calculate how long until a future global time will occur. This last facility would be useful for an application that wants to do something at a specific global time. For example, a common use might be for a node to be able to sleep until the next communication window. If the next window will occur at a particular global time, the node can find out how long it will be until then in local time and then set a timer accordingly so it can wake up and be ready to listen.

All timestamps and clocks were based on the 32768 Hz crystal on the Telos board. The crystal exhibits good short-term stability, which is essential for the FTSP to work properly. However, with a period of about 30.5 µs, it does not support very high-resolution synchronization. An attempt was made to implement the FTSP based on a nominal 1 MHz clock derived from the microcontroller's internal Digitally Controlled Oscillator (DCO), but its short-term stability was too poor, and it could not be used for time synchronization.

IV. RESULTS AND CONCLUSION

Testing of the FTSP implementation indicated very good time synchronization. The tests consisted of a master node and slave nodes connected to a common signal. Once the network was synchronized, the nodes would report the associated global timestamp every time the signal changed state. For each event, the master's timestamp was compared to the slave's timestamp to determine the slave's error. In all cases, the slave node's error was never more than ±2 ticks (61 µs).

It is expected that the more frequently time sync messages are sent, the better the network nodes will be able to maintain synchronization. However, increasing the time between messages did not significantly degrade performance. Testing consisted of four scenarios (A-D) defined by the time sync message frequency and the test duration. Each test ran long

TABLE I: TESTING RESULTS

	A	B	C	D
Message Freq. (sec)	2	10	30	30
Test Duration (min)	2	2	2	120
Average Error (ticks)	0.49	0.61	0.81	0.67
Std. Deviation (ticks)	0.56	0.53	0.48	0.59

enough to provide at least 500 event timestamps for error comparison; test D provided more than 1400 timestamps. The results, shown in Table I, are very good and indicate that highly accurate synchronization could likely be achieved with a high frequency clock source (i.e. several MHz) of adequate stability. Our attempts to demonstrate high-resolution synchronization failed because of the poor quality DCO clock source available on the Telos board.

Coupling the strengths of the FTSP and the master-slave configuration of ZigBee promises to be a profitable means of implementing time synchronization in a WSN. The FTSP can be tailored to require even less from slave sensors while still providing the same degree of reliability and precision. Other peer-to-peer portions of the ZigBee network could continue to use the FTSP in its original form. In the master-slave portions of the network, the master elements can periodically include the time sync information with their regular beacon message. The slaves would not be required to respond and could thus conserve energy. Additionally, they can even sleep most of the time, ignoring most broadcasts from the master and only needing to wake up to hear the time sync broadcast. The precise time synchronization could also provide the added benefit of beacon messages being required even less frequently to keep communications synchronized between the master and slaves. We implemented an application-specific time synchronization protocol using specific ZigBee features in master-slave configuration using the Telos mote platform and the TinyOS development environment. This paper presents the implementation and performance measures collected in an experimental setup.

V. REFERENCES

[1] J. Elson, L. Girod and D. Estrin, “Fine-Grained Network Time Synchronization using Reference Broadcasts,” in *Proceedings of the fifth symposium OSDI ‘02*, pp. 147-163.

[2] J. Elson and D. Estrin. "Time Synchronization for Wireless Sensor Networks,". in *Proceedings of the 2001 International Parallel and Distributed Processing Symposium (IPDPS ‘01)*, pp. 1965-1970.

[3] S. Ganeriwal, R. Kumar, M. B. Srivastava, “Timing-Sync Protocol for Sensor Networks,” in *Proceedings of the 1st International Conference on Embedded Network Sensor Systems (SenSys ’03)*, pp. 138-149.

[4] 802.15.4-2003 *IEEE Standard for Information Technology-Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for Low Rate Wireless Personal Area Networks (LR-WPANS)*, 2003.

[5] M Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The Flooding Time Synchronization Protocol," in *Proceedings of the 2nd International Conference on Embedded Network Sensor Systems (SenSys ’04)*, pp. 39-49.

[6] M Maroti, B. Kusy, G. Simon, and A. Ledeczi, "Robust Multi-Hop Time Synchronization in Sensor Networks," in *Proceedings of the International Conference on Wireless Networks (ICWN ’04)*, Volume 1, pp. 454-460.

[7] D. L. Mills, “Internet Time Synchronization: The Network Time Protocol”*, IEEE Transactions on Communications*, COM 39 no. 10, October 1991, pp. 1482-1493.

[8] W. Ye, J. Heidemann, and D. Estrin. “An energy-efficient MAC protocol for wireless sensor networks,” in *Proceedings of the IEEE Infocom*, 2002, pp. 1567-1576.

[9] Telos platform: http://www.moteiv.com/products-reva.php

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

[11] Vanderbilt’s implementation of the FTSP in TinyOS:, http://cvs.sourceforge.net/viewcvs.py/tinyos/minitasks/02/vu/tos/lib/TimeSync/