An implementation of time synchronization in low-power wireless sensor networks

Conference Paper · March 2011

DOI: 10.1109/SSST.2011.5753777

Emil Jovanov
University of Alabama in Huntsville
183 PUBLICATIONS **6,699** CITATIONS
SEE PROFILE

Aleksandar Milenkovic
University of Alabama in Huntsville
117 PUBLICATIONS **4,092** CITATIONS
SEE PROFILE

An Implementation of Time Synchronization in Low-Power Wireless Sensor Networks

Nason Tackett, Emil Jovanov, Aleksandar Milenković¥
Electrical and Computer Engineering Department
University of Alabama in Huntsville
Huntsville, AL 35899 USA
¥ milenka@uah.edu

***Abstract*—Time synchronization plays an important role in wireless sensor networks, enabling correlation of diverse measurements from distributed sensor nodes, shared channel communication, and energy conservation. When designing time synchronization algorithms for wireless sensor networks, a number of parameters needs to be carefully considered, including precision, processing time overheads, memory requirements, and energy costs. In this paper we describe an implementation of time synchronization on a wireless sensor network custom designed for traffic monitoring applications. Each node consists of a magnetic sensor, a microcontroller, and a low-power Nordic nRF24L01+ radio chip. We describe various aspects of our time synchronization implementation, including hardware interfacing, low-level event time-stamping, time synchronization mechanism, and software implementation. We discuss the results of our experimental study focusing on precision and performance overhead.**

***Index Terms*—Sensor Networks, Synchronization, Nordic.**

I. Introduction

Wireless sensor networks represent an emerging computing platform that blends sensing, computation, and communication to provide a new tool in interfacing with physical environments. Wireless sensor networks consist of a large number of tiny and inexpensive computer platforms that are deeply embedded in their environments. These platforms are capable of sensing the environment, processing information on-board, and communicating with each other and with a network server through multi-hop wireless links. They must reliably operate unattended for extended periods of time (months and years), under stringent resource constraints in energy, communication bandwidth, memory capacity, and processing power.

Time is an important aspect in many applications of wireless sensor networks. Sensor platforms can measure time using their local clocks, driven by oscillators. However, because of random phase shifts and drift rates of oscillators, the local time readings start to differ quickly, causing nodes to lose synchronization. To cope with this issue, wireless sensor networks need to employ time synchronization. Time synchronization plays a critical role in wireless sensor networks enabling: (i) communication protocols based on time-division multiple access; (ii) coordinated wakeups from low-power modes, and (iii) distributed sampling and data aggregation.

A number of time synchronization algorithms for wireless sensor networks have been introduced. Most notable protocols are RBS [1], TSPN [2], and FTSP [3]. The Flooding Time Synchronization Protocol (FTSP) was developed to provide a means for network-wide time synchronization in large wireless sensor networks. Time synchronization is achieved with periodic time synchronization messages that carry a very precise timestamp of when the message was sent (global time). A dynamically elected root broadcasts the time synchronization message. Receiving nodes take their respective local timestamps when the time sync message is received (local time) and rebroadcast the time synchronization messages, thus flooding the network. Comparing the global and local timestamps from the last several time synchronization messages, each node computes a simple linear regression to account for the offset and skew in its local clock from the global clock. Cox et al. [4] introduced a simple implementation of this protocol for Zigbee sensor networks with star topology and demonstrated its operation on a network with Telos platforms [5].

In this paper we describe a new implementation of the time synchronization from [4] that is customized for our originally designed low-power wireless sensor network that targets traffic monitoring applications. We discuss time synchronization in general and our application requirements of interest for time synchronization in Section 2. The wireless sensor network for traffic monitoring consists of a custom-made sensor platform featuring magnetic sensors, a microcontroller, and a Nordic nRF24L01+ radio chip. The nodes are designed for small form factor and very low power consumption. The proposed time synchronization implementation is described in Section 3. It covers various aspects of the implementation: hardware interface, communication protocols, software structures, and time synchronization service functions. Section 4 presents the results of our experimental evaluation, which focuses on accuracy, performance and memory overheads. Finally, Section 5 concludes the paper.

II. Time Synchronization

Time synchronization is a common requirement in wireless sensor networks enabling efficient communication, energy conservation, and collaborative processing. Sensor nodes share radio spectrum and to conserve energy it is advantageous to avoid costly retransmissions of radio packets due to collisions. A common solution is to allocate specific periods of time in which each sensor node is allowed to transmit data. This method is known as time-division multiplexing. Time synchronization is crucial to coordinate timeslots when each node can transmit. Next, time synchronization allows individual sensor nodes to stay in power-down operating modes, thus conserving energy. For example, radio chips can be turned-on only during timeslots reserved for a particular sensor node. Finally, some applications of sensor networks require distributed and collaborative signal processing. Multiple sensor nodes may detect events that need to be precisely time-stamped for later analysis. In such cases it is important that all sensor nodes and gateway share a common and accurate notion of time, regardless of the device type and clock source. Again, time synchronization is instrumental for providing abstractions to support this type of collaborative processing.

Wireless sensor network applications often pose different requirements to time synchronization algorithms. When designing new or evaluating the existing time synchronization methods, several synchronization metrics should be considered. Among these metrics are precision, energy costs, computational resources, memory requirements, and fault tolerance. Thus, we will first discuss characteristics of our wireless sensor network application which targets traffic monitoring with a special focus on requirements relevant to time synchronization.

Our traffic monitoring system consists of multiple wireless sensor nodes deployed on the top of roadways, a gateway, and a centralized server (Fig. 1). Each sensor node consists of a magnetic sensor, an embedded microcontroller, a radio, and an energy source. The magnetic sensor, which measures the earth's magnetic field, tracks disturbances to this field as vehicles pass nearby. To detect a passing vehicle, the microcontroller performs sampling and processing of magnetic sensor signals. The signal processing involves detecting and time-stamping relevant events (e.g., beginning of a disturbance), or even analyzing the shape of disturbances to gain additional insights (e.g., about the size of the vehicle).

The sensor nodes wirelessly send the processed information about relevant events to the gateway. The gateway employs collaborative processing of events streamed from the sensor nodes to get not only statistics related to the number of vehicles on the roadways as a function of time, but also vehicles' speeds and sizes. Key requirements for such a system are a robust design and small form factor for sensor nodes, so that they can be easily deployed on the top of roadways. The small form factor imposes limits on the size of the battery, which in turn translates into limited battery capacity (typically, the battery capacity is directly proportional to its size and weight). In addition, the traffic monitoring system should also provide reliable and autonomous operation for extended periods of time, without frequent battery changes. Consequently, providing a low-power operation is crucial for designing cost-effective and practical traffic monitoring system.

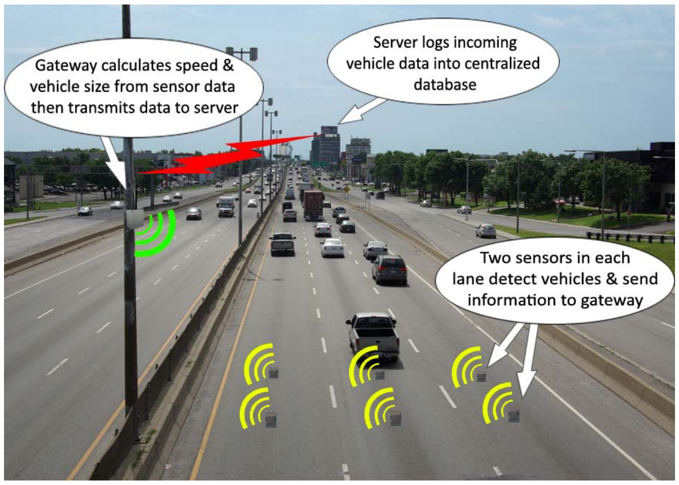
Gateway calculates speed & vehicle size from sensor data then transmits data to server
Server logs incoming vehicle data into centralized database
Two sensors in each lane detect vehicles & send information to gateway

Fig. 1. Traffic monitoring system.

To meet the application requirements for size and low-power operation we opted to design our custom sensor platforms. Typically, a radio interface is the main contributor to the energy consumed on a sensor platform. For example, the current drawn by radio interfaces is 5-10 times larger than the current drawn by a microcontroller [6]. Consequently, we decided to use one of the most energy-efficient radio controllers, a Nordic's nRF24L01+. Thus, our sensor platforms include a Texas Instruments's MSP430 mixed signal system-on-a-chip MSP430F2410 (includes 16-bit processor core with RAM, flash memory, and a rich set of analog and digital peripherals) [7], a Nordic's nRF24L01+ single-chip radio [8], a magnetic sensor [9], and a power source.

Implementing cost-effective and precise time synchronization is instrumental for traffic monitoring applications. It is crucial to support our communication protocol, to coordinate power-down modes of wireless radios, and to enable collaborative processing at the gateway (e.g., in determining vehicle speed from two sensors as illustrated in Fig. 1). Unfortunately, the available solutions for time synchronization were not readily available or applicable to our sensor platform. This motivated us to develop our own implementation of the time synchronization method based on a modified FTSP algorithm proposed by Cox et al. [4].

The gateway is the master node and it periodically sends time synchronization messages that carry a timestamp with global time. The sensor platforms, slave nodes in this case, receive time synchronization messages and timestamp them using their local time. Each slave node maintains its own table with global, local, and difference time stamps for several recent time synchronization messages. The slaves use a simple linear regression to compute the offset and the skew between their local times and the global time. This enables each slave to maintain a very precise estimate of the global time regardless of whether its clock is running faster or slower than the global clock.

III. Implementation of Time Synchronization

Time synchronization implementation in general spans different layers of abstractions, from hardware interfaces, communication protocols, to software structures and functions required to maintain precise time synchronization. In this section we first discuss the hardware interface specific for our sensor platforms and then we discuss network operation and software implementation.

Fig. 2 illustrates the interface between the Nordic nRF24L01+ radio chip and the MS430 microcontroller (top) and a radio packet format (bottom). The microcontroller configures the radio and controls its operation by sending and receiving commands and data via a synchronous peripheral interface (SPI). The radio chip can be configured to operate in different operating modes (e.g., Transmit, Receive, Standby, or Power-down).

The radio chip automatically assembles or disassembles a radio packet that contains a preamble, address, control, variable payload, and CRC field (Fig. 2, bottom). An important part of the interface for time synchronization is the IRQ signal. The radio chip drives this signal and it can be configured to generate an interrupt request on certain events (e.g., when a new packet has been received or when a packet has been sent.)

Our traffic monitoring system employs a star topology with the gateway serving as the network master. The time maintained by the gateway is called global time and each sensor node has its own local time. The gateway periodically broadcasts a time synchronization message that carries information about the global time captured at the master. All sensor nodes, acting as slaves, receive the time synchronization message. In addition, each node captures its respective local time, and updates its time synchronization table that maintains recent (globalTime, localTime, difference) entries. Upon receiving a new time synchronization message, each slave calculates skew and offset parameters using linear regression [3][4] in order to determine local clock drift relative to the clock at the master.

In order to synchronize the master and slaves there must be a fixed point in time that can be referenced by all participating parties without any software-induced delay. While many radios provide a dedicated signal that acts as a start-of-frame delimiter [4], the Nordic radio only has an interrupt request line that can be configured to become active when a radio packet has been sent or received. Luckily, this signal can be utilized for precise time-stamping. We connect this signal to a microcontroller's timer peripheral. The timer peripheral supports hardware capture of the timer counter triggered by an external signal (in our case falling edge of the IRQ signal). This way, a precise

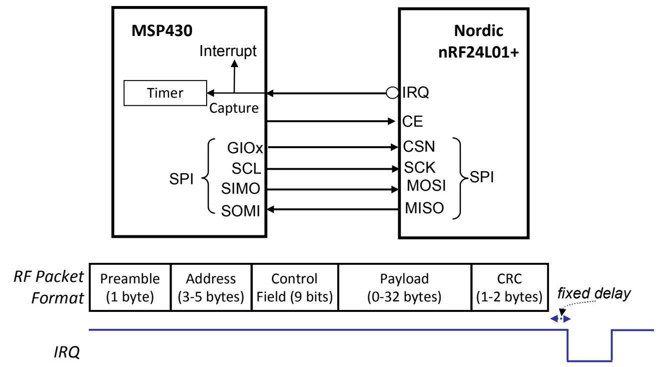

Fig. 2. Interfacing Nordic's nRF24L01+ transceiver (top). An enhanced ShockBurst packet format and IRQ signal (bottom).

time-stamping of the active edge of the interrupt signal can be achieved at the master and all the slaves. Note that the propagation delay of a radio packet is negligible in our case. For example, assuming a distance between the master and a slave of 150 meters, it is 0.5 μs ($150/3*10^8$), which is negligible relative to the timer clock of 30.5 μs. In addition to causing a capture at the timer, the active edge of the IRQ is configured to raise an interrupt request. The corresponding interrupt service routine reads the captured value from the microcontroller's timer peripheral and converts it to the local time. The timer in the MSP430 microcontroller is a 16-bit peripheral. However, our local time variable maintained in software is typically 24-bit or 32-bit long; in general, the length of the time variable is a design parameter.

Fig. 3 illustrates the proposed time synchronization algorithm. The master node periodically broadcasts a time synchronization message. The frequency of time synchronization messages is a design parameter and depends on various factors, including, clock stability, environmental conditions, operating conditions, power and performance overheads required by time synchronization. Let us say that we consider the master sending *i*-th time synchronization message. The payload of the time synchronization message includes its sequence number, *i*, and the timestamp that was captured at the master upon transmission of the previous synchronization message (GT(*i-1*)). The sequence number is required to enable detection of possible lost synchronization messages at each individual slave. The master captures a new time stamp, GT(*i*) upon transmission of the current synchronization message. This timestamp will be included as the payload in the next *(i+1)*-th synchronization message.

All slave nodes maintain a software structure dedicated to synchronization, called time synchronization table. The table includes *n* entries (*n* is a design parameter) that are updated in a round-robin fashion. Each entry in the table includes a local timestamp, a global timestamp, the difference, and a valid flag (Fig. 3). A slave captures its local timestamp when a new time synchronization message has been received (LT(i)). The next entry in the table is updated with the local timestamp LT(i) as illustrated in Fig. 3. This timestamp corresponds to the global timestamp captured at the master, but it will be received in the next time synchronization message. The global timestamp, GT(i-1), captured at the master upon transmission of the previous time synchronization message, is extracted from the current message and the table is updated as shown in Fig. 3. In case of a lost time synchronization message, the slave will invalidate the corresponding entry in the table and update the next entry pointer.

The microcontroller executes several tasks related to time synchronization, including the calculations of the skew and offset parameters. Upon receiving each synchronization message the skew and offset are updated as follows. First, we calculate the offset as the average difference between the global and local timestamps (Eq. 1). We also determine the average local timestamp (Eq. 1). The skew is calculated as shown in Eq. 2 using a classical linear regression. Based on these two parameters a slave can determine a global timestamp GT_x for any local event using the local timestamp LT_x (Eq. 3). So, when a node reports the time when an event has occurred, the global timestamp is calculated and sent to the gateway.

$$offset = \frac{1}{n}\cdot\sum_i Diff_i = \frac{1}{n}\cdot\sum_i (GT_i - LT_i), \quad \overline{LT} = \frac{1}{n}\cdot\sum_i LT_i \qquad \text{Eq. 1}$$

$$skew = \frac{\sum_i (LT_i - \overline{LT})\cdot(Diff_i - offset)}{\sum_i (LT_i - \overline{LT})^2} \qquad \text{Eq. 2}$$

$$GT_x = LT_x + offset + skew\cdot(LT_x - \overline{LT}) \qquad \text{Eq. 3}$$

The time synchronization table needs to be fully or partially filled with valid (LocalTime, GlobalTime, difference) entries before the offset and skew parameters can be accurately calculated. The total number of entries in the time synchronization table and the minimum number of entries needed to calculate the offset and skew are design parameters. With an increase in the number of entries the accuracy of calculations increases too. However, it also increases processing, memory, and energy overheads, which is undesirable in resource-constrained sensor nodes. Our experiments indicate that a minimum of four time synchronization messages is sufficient to calculate the offset and skew, allowing for fast synchronization with accuracy of ±2 timer clocks.

We have extended our time synchronization mechanism to include an accuracy check procedure. The slave nodes verify the accuracy of time synchronization by comparing each of the global times in the synchronization table (received from the master) with the corresponding calculated global times using Eq. 3. If each of the calculated global times are equal to the respective global times, then the device exhibits good time synchronization. When a slave node reaches successful time synchronization, it continues to recalculate the offset and skew each time a new time synchronization message is received. However, if the average difference between global time and the calculated global time are more than one clock cycle off (this is a design parameter), the most recent values for the offset and skew parameters are not used. Instead, the slave requests fast synchronization and continues to operate using the last good offset and skew parameters, until new stable values are found.

In addition to time synchronization tasks taken in a steady state, our implementation involves handling exceptional

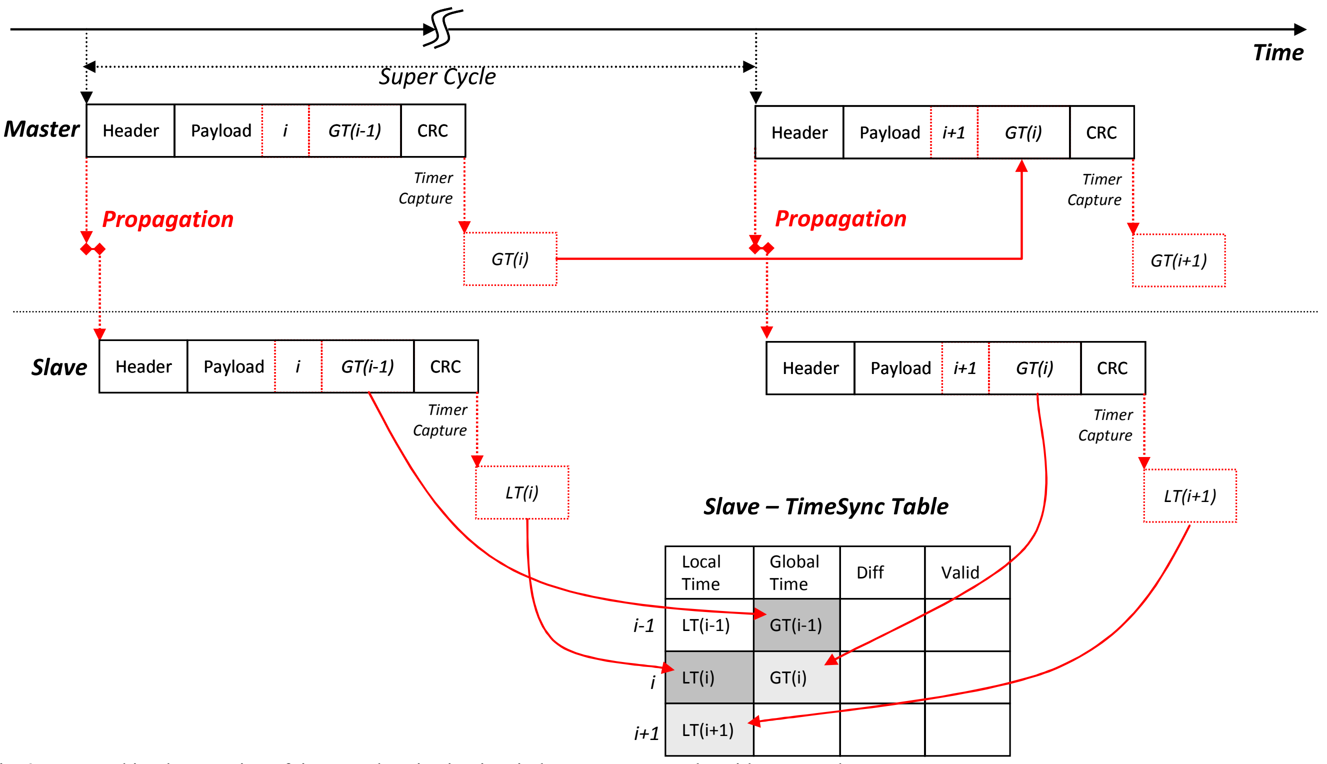

Fig. 3. Proposed implementation of time synchronization in wireless sensor networks with star topology.

situations. In the remaining of this section we discuss time synchronization activities taken when a new slave node joins the network, when the master node reboots, and when the time variables rollover.

Initializing the time synchronization table when a slave joins the network may take a relatively long time, especially in cases of infrequent broadcasts of time synchronization messages. To expedite this process, we allowed for fast synchronization. When a slave node comes online, it sends a message to the master requesting fast synchronization. When the master receives such a request, it notifies all the slaves that time synchronization messages will be sent at a faster rate. Note: in case that the slaves are in a power-down mode, this notification is carried out during regular time synchronization time slot. When the slave who initiated fast synchronization achieves it, it sends a fast synchronization termination request. The master then notifies all the slaves about switching to regular time synchronization period.

Another boundary condition is when the master (network gateway) undergoes a reboot. The master sends out a “boot announce” message to inform all the slaves that the existing time synchronization tables should be invalidated. The slaves continue to use the last “good” offset and skew parameters until enough time synchronization messages have been received. The slaves use the payload of the first time synchronization message after the “boot announce” message to set their local time close to the new global time. This way, the offset parameters at each slave are kept as close to zero as possible.

We have chosen 32-bit unsigned integers for local and global time variables. The microcontroller uses a 16-bit timer peripheral though, and the interrupt service routine caused by the timer’s counter rollover or an external event is used to update the local time variable. We use a stable external 32 KHz clock source as the timer’s clock source. This clock proves to be sufficient for our application. A small example is used to support this claim. Let us consider two sensor nodes placed on the top of the road at the distance of 2 meters from each other. A car moving 120 km/h (33.33 m/s) will travel the distance between the sensors in 60 ms; the same car moving 121 km/h (33.61 m/s) will travel this distance in 59.504 ms. The difference between these two travel times equates roughly to 16 timer ticks, which is deemed sufficient for our application. Please note that the microcontroller features an internal digitally-controlled oscillator capable of running at up to 16 MHz clock frequency and this clock could be used in applications requiring higher clock resolution.

A clock roll-over exception occurs when either the master global clock or the slave’s local clock reaches the top of its 32-bit range and resets back to zero. With our clock source it is expected to happen every 18 hours. This introduces a problem for the time synchronization offset and skew calculations. To properly handle this exception, a time rollover detection process is introduced. When it is detected that either the global clock or the local clock rolled back over to zero, the time table is flushed, and the last good values for the offset and skew are used until the table is refilled and successful synchronization is achieved with the new values.

IV. Experimental Evaluation

The goal of our experimental evaluation is to test the accuracy of the proposed implementation of time synchronization. We also discuss the processing overhead associated with time synchronization service functions as well as some implementation implications. Finally, the time synchronization is tested in the context of the target traffic monitoring application.

Our testing setup consists of a master node and a slave node. To test time synchronization we connect a 4 Hz square-wave signal from a function generator to both the master and the slave. The slave captures the rising edge of the signal, timestamps it using its local time, coverts the local time to global time using the skew and offset parameters, and wirelessly sends the global timestamp to the master. The master captures the rising edge of the test signal too, timestamps it using its local time variable (global in the network), receives the timestamp from the slave, and forwards both timestamps to a workstation through a serial link. The workstation runs an application that logs the slave’s and master’s timestamps. The experiment is repeated by varying several parameters, including, time synchronization period, the time synchronization table size, and local time length.

First, let us discuss the issue of the type and length of time variables and the skew and offset parameters. Local and global time variables maintained by each node were originally unsigned 32-bit numbers. The offset and skew parameters are initially defined as single-precision floating-point numbers. The arithmetic operations described in Eq. 1 – Eq. 3 are performed on single-precision numbers. However, in the case when the time variables become larger than 2^{24}, some significant bits are lost in conversions of 32-bit unsigned integers into floating point numbers (mantissa is 23-bit, plus an implicit additional bit in single-precision representation), causing incorrect results and failure of synchronization. To remedy this problem we may limit the range of time variables to be between 0 and 2^{24}. The software is modified to detect these new rollover conditions. It should be noted that the chosen microcontroller does not include native support for floating-point instructions. Rather, these operations are emulated in software, and thus tend to take a lot of processor clock cycles.

An alternative approach is to use a full 32-bit range for time variables, but to perform operations using 64-bit double-precision numbers (the skew is a 64-bit number). While this approach allows a full 32-bit range of time variables (0 to 2^{32}-1), it significantly lengthens the time required for these operations.

Table I shows the number of clock cycles needed to calculate the offset and skew parameters as a function of the time synchronization table size (4, 8, and 16 entries) and the size of the skew and offset parameters. The column local2global shows the number of clock cycles needed to covert a local time stamp into a global time stamp (operation described in Eq. 3).

The results indicate that the use of double-precision arithmetic almost doubles the overhead needed to calculate the offset and skew parameters. Thus, it is beneficial to use the single-precision calculations and limit the range of time variables to 24 bits.

TABLE I. TIME SYNCHRONIZATION OVERHEAD.

	local2global	Time Synchronization Table Size		
Parameters type/size		4	8	16
64-bit float	*56*	*4431*	*7603*	*14039*
32-bit float	*50*	*2525*	*4013*	*7021*

Table II shows the result of experimental evaluation of time synchronization for an implementation using 24-bit time variables and single-precision arithmetic, as a function of the time synchronization period. The table size is fixed to 8 entries. We consider time synchronization periods of 8, 16, and 32 seconds. We measure the average difference or error in timer clock cycles (jiffy clocks) between the reported master and slave global times (AvgDiff), the standard deviation (StdDev) of the difference, and the variance of the difference (Variance). We also report the range of maximum differences (negative differences indicate situations when the slaves reported global clock is behind the master's global clock, and positive when the slave reported time is higher than the master's global clock). The last column shows the percentage of time the slave spent in a fast synchronization mode. We can see that the proposed mechanism remains stable even when we reduce the frequency of time synchronization messages from 8 seconds to 32 seconds. The smallest average difference is observed when time synchronization period is set to 16 seconds and it is only -0.188 jiffy clock. The standard deviation is also relatively small, somewhat better for more frequent messages. The upper bounds indicate that the maximum deviations in reported times do not exceed 2 jiffy clocks for synchronization periods of 8 and 16 seconds and 3 jiffy clocks when synchronization period is 32 seconds. The percentage of time spent in fast time synchronization when time synchronization messages are sent every 32 seconds is relatively small at 3.6%.

TABLE II. TIME SYNCHRONIZATION EVALUATION
AS A FUNCTION OF TIME SYNCHRONIZATION PERIOD.

Message Period	AvgDiff	StdDev	Variance	-	+	% Fast Sync
8	0.286	0.527	0.278	-2	2	6.77
16	-0.188	0.606	0.367	-2	1	4.87
32	-0.541	0.703	0.494	-3	1	3.58

Table III shows the result of experimental evaluation of time synchronization for an implementation using 32-bit time variables and double-precision arithmetic, as a function of the time synchronization table size. We can see that somewhat surprisingly smaller table sizes tend to give better results (smaller average difference and standard deviation). This is desirable because smaller table reduces both memory required for the time synchronization table and performance overhead for calculation of the offset and skew parameters.

TABLE III. TIME SYNCHRONIZATION EVALUATION
AS A FUNCTION OF TABLE SIZE.

Table Size	AvgDiff	StdDev	Variance	-	+	% Fast Sync
4	0.111	0.555	0.308	-2	2	6.40
8	-0.188	0.606	0.367	-2	1	4.87
16	0.124	0.612	0.375	-2	2	3.92

V. CONCLUSION

This paper describes an implementation of time synchronization protocol on custom-designed wireless sensor network platforms for traffic monitoring applications. To the best of our knowledge we offer the first practical time synchronization on sensor platforms based on Nordic's low-power radios. The paper describes all the layers of interest for time synchronization, from the hardware interface to data structures and software procedures. In addition, we offer results of our experimental evaluation that confirms that the proposed mechanism fully meets our application requirements.

REFERENCES

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

[2] 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.

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

[4] D. Cox, E. Jovanov, A. Milenkovic, "Time Synchronization for ZigBee Networks," in *Proc. of the 37th IEEE Southeastern Symposium on System Theory (SSST'05)*, Tuskegee, AL, March 2005, pp. 135-138.

[5] J. Polastre, R. Szewczyk, and D. Culler. "Telos: Enabling ultra-low power wireless research," in *Proc. of the 4th International Conference on Information Processing in Sensor Networks: Special track on Platform Tools and Design Methods for Network Embedded Sensors (IPSN'05)*, April 25-27, 2005, pp. 364-369.

[6] A. Milenkovic, M. Milenkovic, E. Jovanov, D. Hite, "An Environment for Runtime Power Monitoring of Wireless Sensor Network Platforms," in *Proc. of the 37th IEEE Southeastern Symposium on System Theory (SSST'05)*, Tuskegee, AL, March 2005, pp. 406-410.

[7] Texas Instrument MSP430 mixed signal system-on-a-chip: http://www.ti.com/home_p_micro

[8] Nordic Semiconductor nRF24L01+ radio: http://www.nordicsemi.com

[9] Honeywell HMC1052L magnetic sensor: http://www.honeywell.com