

# On the Interaction of Clocks, Power, and Synchronization in Duty-Cycled Embedded Sensor Nodes

THOMAS SCHMID, ROY SHEA, ZAINUL CHARBIWALA, JONATHAN FRIEDMAN,  
MANI B. SRIVASTAVA

Department of Electrical Engineering  
University of California, Los Angeles

and

YOUNG H. CHO

The Computer Network Division  
University of Southern California  
Information Sciences Institute

---

The efficiency of the time synchronization service in wireless sensor networks is tightly connected to the design of the radio, the quality of the clocking hardware, and the synchronization algorithm employed. While improvements can be made on all levels of the system, over the last few years, most work has focused on the algorithmic level to minimize message exchange and in radio architectures to provide accurate time-stamping mechanisms. Surprisingly, the influences of the underlying clock system and its impact on the overall synchronization accuracy has largely been unstudied.

In this work, we investigate the impact of the clocking subsystem on the time synchronization service and address, in particular, the influence of changes in environmental temperature on clock drift in highly duty-cycled wireless sensor nodes. We also develop formulas that help the system architect choose the optimal resynchronization period to achieve a given synchronization accuracy. We find that the synchronization accuracy has a two region behavior. In the first region, the synchronization accuracy is limited by quantization error, while in the second region changes in environmental temperature impact the achievable accuracy. We verify our analytic results in simulation and real hardware experiments.

Categories and Subject Descriptors: B.m [**Hardware**]: Miscellaneous

General Terms: Experimentation, Measurement

Additional Key Words and Phrases: Oscillator, Clocks, Duty-Cycling, Time Synchronization, Temperature Effects

---

Author's address: Thomas Schmid, Roy Shea, Zainul Charbiwala, Jonathan Friedman, Mani B. Srivastava; Networked and Embedded Systems Lab; University of California, Los Angeles; 420 Westwood Plaza; Los Angeles, CA 90095; email:thomas.schmid,zainul,jf8,mbsucla.edu, rshea@cs.ucla.edu

Young H. Cho; Division 7, Networks; 4676 Admiralty Way, Suite 1147; Marina del Rey, CA 90292; email:youngcho@isi.edu

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© 20YY ACM 0362-5915/20YY/0300-0111 \$5.00

## 1. INTRODUCTION

Accurate time keeping enables synchronization, which is a vital prerequisite to coordination, security, and detection in distributed systems. The desire for synchrony and the limited access to energy are exacerbated when the distributed system uses wireless links. For many years, researchers have investigated different aspects of time synchronization in wireless embedded systems. A survey of many of these protocols can be found in [Sundararaman et al. 2005].

Current state of the art time synchronization protocols in sensor networks [Maróti et al. 2004; Sallai et al. 2006] achieve an accuracy between 1 to 10 *microseconds*. This level of accuracy suffices for many environmental monitoring applications and acoustic localization systems (e.g. a sniper localization system [Simon et al. 2004]). However, many systems, especially those used in the wired network community need significantly more accuracy for control and measurement. The Precision Time Protocol, IEEE 1588 [Eidson 2006], was developed for accuracies below 100 *nanoseconds* and was standardized in 2002<sup>1</sup> because NTP [Mills 1991], the de facto standard to synchronize computers over the Internet, did not provide enough accuracy.

To achieve the required accuracy, any IEEE 1588 implementation will require resynchronization almost every second, due to clock uncertainties, to remain within specification and even more frequently for accuracies beyond the baseline. One common technique to deal with this uncertainty is to estimate the frequency error of the local clock. Two aspects, manufacturing inaccuracies and changes in environmental temperature, are the two major contributors to this uncertainty. While manufacturing imprecision is static over the lifetime of a component, changes in environmental temperature require frequent recalibration. In indoors wired systems, as is often the case for measurement and control instruments, the environmental temperature changes are slow, and thus adaptation is easily achieved. Additionally, the systems usually are not as bandwidth constrained as their wireless counterparts, and thus frequent communication, even at one second intervals do not cause issues.

The manufacturing industry is increasingly interested in using IEEE 1588 over wireless networks. However, there is currently no implementation available for IEEE 1588 over wireless channels. Only preliminary studies have been conducted on the feasibility of such a protocol [Cooklev et al. 2007]. Problems occur because uncertainties are introduced by the wireless channel. Deep fading, interference, unpredictable latencies due to the broadcast nature of a wireless channel, and higher bit error rates than in wired networks all complicate synchronization efforts.

Implementing IEEE 1588 in sensor networks faces yet another issue due to the frequent resynchronization periods – power consumption. In sensor networks, where energy is scarce and communication is the largest component of the energy budget, the frequent communication exchanges are clearly infeasible.

In most earlier research, accuracy was the sole driving application. The energy

---

<sup>1</sup>The latest revision is from 2008, which addressed several performance enhancements.

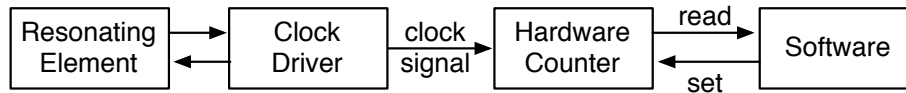


Fig. 1. Basic block diagram of a clock circuit and associated timer hardware. The clock driver excites the resonating element that works as a filter and eventually resonates at a frequency. The hardware counter uses that signal to increment a counting register at a regular interval. Software can use this hardware counter for timers and time measurement.

consumption of these algorithms has largely been ignored. At the best case, the algorithms compared the number of messages sent per synchronization exchange between the nodes. While this is an important measure, other components of the hardware platform start to become much more relevant in the power budget as communication overhead gets reduced. These overheads are particularly pronounced in the extremely low duty-cycles wireless sensor networks try to achieve. In such a scenario, a high precision and high frequency clock can quickly become the most significant energy consumer during sleep times, thus invalidating any power gains made in synchronization accuracy.

This work investigates the interplay of the choices in clock subsystem, synchronization accuracy, and power in duty cycled wireless embedded systems. While the accuracy of a time synchronization algorithm is very important, the link between the source of time on an embedded system and the achievable accuracy has been poorly studied. Assumptions like constant clock skew during resynchronization are commonplace in these algorithms. In this work we show the limits of this assumption and where they start to break down. These are important considerations for sensor network system architects who want to reduce the power consumption of a synchronization algorithm by increasing the time between resynchronization attempts.

## 2. CLOCKING TERMINOLOGY

Time in embedded systems is usually kept by a specialized sub-system illustrated in Figure 1. A periodic *clock signal* lies at the core of the system. The clock signal increments a hardware counter every  $1/f$  seconds, where  $f$  is called the frequency of the clock signal. Therefore, at any time  $t$  since the  $n$ -bit counter was reset, the counter reads  $c(t) = \lfloor f \cdot t \rfloor \bmod 2^n$ . The floor operator  $\lfloor \cdot \rfloor$  comes into play due to the digital nature of the hardware counter. The  $1/f$  rate at which the counter is incremented is called the *resolution*. However, high resolution is not useful if the software cannot read the counter at that speed. Therefore, the smallest increment at which an application can read the counter is called *precision*. Finally, the counter is typically set to an international time calendar, e.g. UTC. The *accuracy* determines how true the counter holds to that calendar.

The clock signal is a periodic signal with some nominal frequency  $f_0$ . Every clock signal will deviate from its intended nominal frequency for both dynamic (environmental changes including pressure, temperature, acceleration) and static (imprecision in its manufacture) reasons. This deviation is termed *frequency error*

(or inversely *frequency stability*) defined as  $f_{\text{error}}(t) = f_0 - f(t)$ , where  $f(t)$  is the frequency of the clock signal at time  $t$ . In general, this error is very small and is commonly expressed in a unitless quantity parts per million (ppm) derived from Equation (1).

$$f_e(t) = \frac{f(t) - f_0}{f_0} \cdot 10^6 \quad (1)$$

The frequency error of a clock signal changes over time. This change, called *frequency drift*, can be classified into two categories: short term (seconds to days) and long term (weeks to years). The observable short term changes are due to changes in the environment. Rapid acceleration, like in rockets, or changes of temperature and pressure on the circuit can affect the frequency error by tens to hundreds of ppm. Long term changes occur as the clock ages. The physical stress induced on components over time can change their electrical properties, and thus introduces a frequency error of the order of a few ppm every year.

The behavior of the frequency error largely depends on the underlying technology used to generate the clock signal itself. In general, two components are necessary to create a clock signal, a *resonating element*, and a *clock driver*. The resonating element is responsible to create an oscillation. The element by itself however does not sustain the oscillation and a clock drive circuit is necessary to initiate and sustain that oscillation. More specifically, it provides both feedback (must meet the Barkhausen [Haque and Cox 2004] criteria with  $gain \geq 1$ , total loop phase  $\geq 2\pi$ ) and isolation to the resonator. Although the ideal driver varies by oscillator technology, CMOS buffers or inverters are generally well-suited since they have high input impedance (except w.r.t. the low impedance of quartz crystals excited into series resonance), high gain, and high bandwidth.

The choice of inverter, buffered or unbuffered, and the number of inverters in the feedback loop influences the characteristics of the clock signal, its drive strength, and the total power consumption of the clocking circuit. Often times, the clock driver is either integrated with the resonating element (usually called an oscillator) or it resides within the microprocessor. In the first case, the driver will have high drive strength in order to deliver a high quality clock signal to many different digital components, though it will consume a lot of power. In the second case, the clock driver is matched to the internal clock signal network of the microprocessor, and thus is optimized for energy consumption. For example, a clock oscillator from Abracon at 16MHz drains about 8.0 mW while clocking a Texas Instrument MSP430 Microprocessor. This is about 20% more power than if the MSP430 is clocked directly using a resonating element (Citizen 16MHz crystal) and its internal clock driver, which we measured at 6.1 mW.

The frequency error introduced by changes in the clock driver are usually not significant. It is in large part due to physical changes in the resonating element that the frequency of the clock signal changes over time. Therefore it is important to know the different technologies that can be used as resonating elements, since each of them has different performance characteristics. Table I summarizes the characteristics of the most common resonating elements, and we refer the interested reader to [Schmid et al. 2009] for a more detailed discussion.

Table I. Comparison of different Resonating Elements

Type	Stability	Power	Cost
LC/RC	1000's of ppm	Low	Cents to free
Ring Oscillator	1000's of ppm	Low	Cents to free
Crystal	10's of ppm <1ppm if controlled	Low - High Freq. Dependent	10's of cents to Dollars
Crystal Oscillator	<1ppm to 10's of ppm	Med - High	Dollars to 10's of Dollars
MEMS Resonator	10's to 100's of ppm	Med - High	10's of Cents to Dollars

### 3. UNDERSTANDING POWER CONSUMPTION IN DUTY-CYCLED DEVICES

Duty cycling has become a critical technique to minimize the power consumption in wireless embedded sensing devices. The intuition behind this is clear: keep hardware in a low power sleep state except during the infrequent instances when the hardware is needed. In many environments this allows even the processor to be put into a low power state for extended periods of time while only an external clock tracks time to trigger a later wakeup.

Recent work has identified clock stability as a limiting factor for the duty cycling that is possible in networks of devices using scheduled communication. When duty cycling in scheduled communication systems, it is important that the communicating nodes wake up at the correct time so that they can talk to each other. Less stable clocks require nodes to resynchronize more frequently to account for clock frequency error. For example, Dutta showed in [Dutta et al. 2007] that the lower bound of a clock with a stability of  $\pm 50$ ppm is a duty cycle of 0.01% for a scheduled communication MAC protocol. It thus appears that more stable clocks are necessary in order to improve the duty cycling capabilities of embedded systems. The following formulation gives an in-depth understand of this interplay.

#### 3.1 Clock Stability and Duty Cycling

The following formalization examines the average power consumption of two systems,  $M$  and  $N$ , using different clocks. We assume both systems have the same sleep power consumption  $P_s$  and active power consumption  $P_a$ . Each node has a local clock source with frequency stability  $s_M$  and  $s_N$  (in  $Hz/Hz$ ) that consume power  $P_{cM}$  and  $P_{cN}$ , respectively. Additionally, we assume that both platforms have the same duty cycle ratio  $DC$  between sleep time  $T_s$  and active time  $T_a$ . However, in order to communicate with their peers, both systems include a guard time that is proportional to their local clock source's frequency stability  $T_{gX} = 2 \cdot T_s \cdot s_X$ . This guard time allows the nodes to compensate for the drift in their clock frequency while asleep by starting the synchronization process early enough to ensure that both nodes are awake when the active period starts. The guard time is set by the resynchronization period, which, in this case, is the sleep period since we assume nodes will resynchronize when they are awake. Note that resynchronization in this case comes for free, since it implicitly happens whenever a message gets transmitted from  $M$  to  $N$ , or vice-versa.

The average power consumption of this system is simply the sum of the power consumed in sleep mode and the power consumed in awake mode, normalized by

the total time the system is on:

$$P_X = \frac{T_s \cdot (P_s + P_{cX}) + (T_{gX} + T_a) \cdot (P_a + P_{cX})}{T_s + T_a + T_{gX}}, \quad (2)$$

where the subscripted  $X$  is either  $M$  or  $N$ . Let us also assume that, without loss of generality,  $N$  is a more stable clock so that  $s_M > s_N$  and consequently  $P_{cM} < P_{cN}$ . In order for system  $N$  to be more efficient than system  $M$  we have to show that

$$P_M > P_N, \quad (3)$$

and thus

$$\begin{aligned} & \frac{T_s \cdot (P_s + P_{cM}) + (T_{gM} + T_a) \cdot (P_a + P_{cM})}{T_s + T_a + T_{gM}} > \\ & \frac{T_s \cdot (P_s + P_{cN}) + (T_{gN} + T_a) \cdot (P_a + P_{cN})}{T_s + T_a + T_{gN}}. \end{aligned} \quad (4)$$

Quartz crystals are commonly used in embedded systems, are inexpensive, and are a representative clocking mechanism. For comparison, we now assume that the clock of system  $M$  is a quartz crystal. These crystals consume very little power, and thus  $P_{cM} \sim 0$ . This assumption is justified by our precise numerical analysis presented in Section 3.2. Using the assumption that  $P_{cM} \sim 0$ , Equation (4) can be simplified to

$$P_{cN} < \frac{2 \cdot T_s^2 \cdot (P_a - P_s) \cdot (s_M - s_N)}{(T_s \cdot (1 + 2s_M) + T_a)(T_s \cdot (1 + 2s_N) + T_a)}, \quad (5)$$

where  $T_{gM}$  and  $T_{gN}$  are replaced by their respective definitions. By definition, the duty cycle of the system is  $DC = T_a / (T_a + T_s)$  assuming that the guard bands are small with respect to the active time  $T_a$ . Since clock stability is small (usually measured in *ppm*) with  $s_M, s_N \ll 1$ , the relation can be further simplified to

$$P_{cN} < 2 \cdot [1 - DC]^2 \cdot (P_a - P_s) \cdot (s_M - s_N). \quad (6)$$

This relation shows that for system  $N$  to be more energy efficient than system  $M$ , its clock power consumption cannot exceed a threshold dependent on the duty cycle, the difference in active and sleep power consumption and the difference in their clock stability. If we further assume that  $DC \ll 1$ ,  $P_s \rightarrow 0$ , and  $s_M \gg s_N$  a final simple relation is reached:

$$P_{cN} < 2 \cdot P_a \cdot s_M. \quad (7)$$

This reveals that for system  $N$  to be more energy efficient than system  $M$ , the clock system used in system  $N$  must use less power than the active power consumption multiplied by twice the precision of system  $M$ 's clock stability. For example, if we assume that  $P_a = 100 \text{ mW}^2$ ,  $s_M = 50 \text{ ppm}$ , and  $s_N = 1 \text{ ppm}$  (in order to satisfy  $s_M \gg s_N$ ) then  $P_{cN} < 10 \text{ } \mu\text{W}$ .

<sup>2</sup>This is a typical value for sensor network platforms like the Epic [Dutta et al. 2008] sensor network platform.

### 3.2 Adding Time Precision

The formalization provided disregarded clock resolution that prevents arbitrary sleep periods due to the discretization of time by the clock ticks. Clock resolution can be easily added into our formalization. The only terms that change are the guard bands. Now the device must wake up  $1/f_X$  seconds earlier in order to guarantee that the node is awake at the correct time. Thus, the guard bands become  $T_{gX} = 2 \cdot T_s \cdot s_X + 1/f_X$ .

Our formalization also made a number of assumptions to simplify the resulting inequality presented in Equation (6) that must be verified. We add time precision to Equation (4) and numerically solve the equation to verify our formalization. We use the same parameters as in the last example, representing a typical wireless sensor network platform:  $P_a = 100$  mW,  $s_M = 50$  ppm, and  $s_N = 1$  ppm. We further add  $DC = 0.01$ ,  $P_s = 5$   $\mu$ W,  $P_{cM} = 10$   $\mu$ W,  $f_M = 32.768$  kHz, and  $f_N = 8$  MHz. Numerically solving Equation (4) extended with clock guard bands taking into account clock resolution reveals the inequality  $P_{cN} \leq 10.7$   $\mu$ W. This is nearly identical to the result from our formalization. We conclude that for guard band reduction, there is no incentive to increase the clock resolution beyond the very common 32 kHz tuning fork crystal clocks.

### 3.3 Discussion

In research, low-power and high stability clocks have been presented [Aebischer et al. 1997], however none of them are readily available despite the more than 10 years of their existence. One currently available product that partially fits into this category is the MAXIM DS32kHz, a 32 kHz TCXO. Its frequency stability is  $\pm 2$  ppm over  $0^\circ\text{C}$  to  $40^\circ\text{C}$  and it drains in its normal dual-supply mode<sup>3</sup> typically 750  $\mu$ W. While this is well above the calculated limit of 10  $\mu$ W, this particular TCXO has a single supply mode in which it can drain as low as 6  $\mu$ W. This is only slightly higher than a regular 32 kHz crystal, and thus a duty-cycle improvement can be made by employing this device.

So why are not all the sensor network platforms equipped with such a clocking system? There are two potential reasons: unit cost and size. The TCXO costs 6 times the price of a regular 32 kHz crystal<sup>4</sup>. Perhaps even more important is that its physical size is many times that of a crystal. Thus it increases the PCB size and indirectly adds addition cost to production.

One disadvantage of this analysis is that it does not consider the inherent communication capabilities of a sensor network for clock calibration. Using communication, a node can time synchronize, and thus calibrate for its clock frequency error. [Maróti et al. 2004] indirectly showed how accurate a node can estimate its current frequency error. In an experiment where they stopped the synchronization process and then continued to collect time accuracy measurements, they showed that over 30 minutes the synchronization accuracy changed from 5  $\mu$ s to 50  $\mu$ s. This corresponds to a frequency error of 0.025 ppm for this particular setup! The problem is

<sup>3</sup>A separate battery power can provide a backup in case the normal power drops below operation voltages.

<sup>4</sup>The DS32kHz costs US\$3.096 @ 1k units, vs. US\$0.462 @ 1k units for a ECS 32 kHz crystal on December 2009.

that there does not exist any study on how the synchronization process and clock source interact with each other, and thus we do not know how the frequency error estimation behaves in other situations. In the following section, we will develop a theory that explains this relation and allows us to show how time synchronization can efficiently save energy in duty-cycled systems.

#### 4. CLOCK-CHARACTERISTIC IMPACTS ON RESYNCHRONIZATION RATE

An isolated clock can not improve or correct its accuracy by itself. As soon as we have a network of clocks with communication capabilities, clocks can use other clocks to verify their accuracy, and possibly increase their stability. Many time synchronization protocols, like [Maróti et al. 2004; Ganeriwal et al. 2005; Elson et al. 2002], showed that high time synchrony can be achieved by a group of nodes. The general approach is to estimate the current frequency error between the nodes using message time-stamping and different estimation techniques. One common assumption of these protocols is that the temperature, and thus the frequency error, is constant over the time intervals these synchronization protocols operate on.

The remainder of this paper investigates the impact of temperature on the frequency error. This section shows that, depending on the system clock characteristics, frequency error can become a problem. This problem is especially pronounced if one wants to increase the sleep periods between resynchronization points to minimize energy consumption. The following analysis sheds light onto the decision of how often a time synchronization protocol has to resynchronize in order to stay within a specific time synchronization error. It also shows how accurate a node can estimate its frequency error, given certain clock characteristics and resynchronization intervals.

##### 4.1 New Terminology

The biggest contributor to frequency change is the change in environmental temperature. We deliberately left out effects of temperature on the frequency error in Section 2 in order to keep the definitions easy to understand. In reality, the frequency  $f$  will change over time. The temperature dependent frequency can be expressed using the nominal frequency  $f_0$  as

$$f(t) = (1 + \delta(t)) \cdot f_0, \quad (8)$$

where  $\delta(t)$  is the frequency error over time and can be expressed as  $\delta(t) = g(\kappa(t))$ . Here,  $g(\cdot)$  is a function defined by the resonating element<sup>5</sup>, and  $\kappa(t)$  is the temperature at time  $t$ . Given this new definition, the counter on a system reads at any time  $t$  since the counter was reset

$$c(t) = \left\lfloor \int_0^t f(\tau) d\tau \right\rfloor = \left\lfloor f_0 \cdot \left( t + \int_0^t g(\kappa(\tau)) d\tau \right) \right\rfloor. \quad (9)$$

<sup>5</sup>For an AT-Cut crystal,  $g(\cdot)$  is a cubic curve, or for a tuning fork crystal,  $g(\cdot)$  is an inverse quadratic curve.

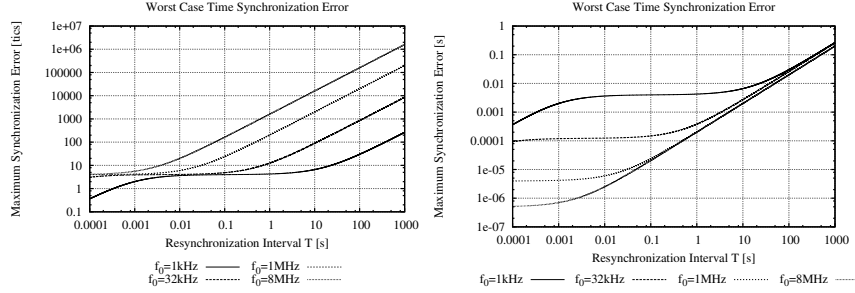


Fig. 2. Worst case time synchronization error for four different clock speeds. The two low frequency clocks are tuning fork crystals with a maximum frequency error of 120 ppm, and the high frequency clocks are AT-cut crystals with a maximum frequency error of  $\pm 50$  ppm.

## 4.2 Time Synchronization Error

We assume a simple two way time synchronization protocol, like IEEE 1588, for our error analysis. More precisely, we consider a two node network, where one node is the master and the second node is the slave. We assume that the master clock has perfect time and that the slave node tries to synchronize to the master node's time. The counter on the master node will read  $c_0(t) = \lfloor f_0 \cdot t + \varphi \rfloor$ , where  $\varphi$  is a random phase offset between  $[0, 1)$ . To make the analysis simpler, we assume that there are no time-stamping errors. We will relax this assumption later on.

The master and slave node regularly exchange time synchronization messages that contain precise time-stamping information. Using these timestamps, the slave node can accurately estimate the current offset between its local clock and the master clock. The slave node estimates its current frequency error using the two last offset measurements exchanged with the master node. Assuming that these two messages were  $T$  seconds apart, the slave node can estimate the local frequency error using

$$\delta_Q = \frac{c(-T) - c(0)}{c_0(-T) - c_0(0)} - 1 = \frac{\lfloor f_0 \cdot T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau \rfloor}{\lfloor f_0 \cdot T + \varphi \rfloor} - 1. \quad (10)$$

Using this frequency error estimate, the synchronization error between  $t = 0$  and the next time synchronization exchange at  $t = T$  is

$$\begin{aligned} \varepsilon_Q(t) &= (c(t) - \delta_Q \cdot c(t)) - c_0(t) \\ &= c(t) - \left( \frac{\lfloor f_0 \cdot T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau \rfloor}{c_0(T)} - 1 \right) \cdot c(t) - c_0(t) \\ &= 2 \cdot c(t) - \frac{\lfloor f_0 \cdot T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau \rfloor}{c_0(T)} \cdot c(t) - c_0(t). \end{aligned} \quad (11)$$

**4.2.1 Bounding the Time Synchronization Error.** We can not find a closed form solution for Equation (11) due to the non-linearity of the floor operator and the unpredictability of the temperature. However, we can find bounds on the maximum of the time synchronization error.

The fractional part function is defined as

$$\{x\} = x - \lfloor x \rfloor, \quad (12)$$

for all  $x$ ,  $0 \leq \{x\} < 1$ . Thus, we can write the floor operator as  $\lfloor x \rfloor = x - \{x\}$ . Substituting the fractional parts with the variable  $\nu_x$  and expanding the counter values  $c(t)$  to their definition, the resynchronization error from Equation (11) becomes

$$\begin{aligned} \varepsilon_Q(t) = & \\ & 2 \cdot (f_0 t + f_0 \int_0^t g(\kappa(\tau)) d\tau - \nu_3) \\ & - \frac{f_0 T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau - \nu_1}{f_0 T + \varphi + \nu_2} \cdot (f_0 t + f_0 \int_0^t g(\kappa(\tau)) d\tau - \nu_3) \\ & - (f_0 T + \varphi - \nu_2). \end{aligned} \quad (13)$$

We can now bound the time synchronization error

$$\begin{aligned} \varepsilon_Q(t) \leq & \max(\varepsilon_Q(t)) \\ \leq & \max \left( 2 \cdot (f_0 t + f_0 \int_0^t g(\kappa(\tau)) d\tau - \nu_3) \right) \\ & - \min \left( \frac{f_0 T + f_0 \int_{-T}^0 g(\kappa(\tau)) d\tau - \nu_1}{f_0 T + \varphi + \nu_2} \cdot (f_0 t + f_0 \int_0^t g(\kappa(\tau)) d\tau - \nu_3) \right) \\ & - \min(f_0 T + \varphi - \nu_2). \end{aligned} \quad (14)$$

From the definition, we know that  $\min(\nu_x) = 0$ , and  $\max(\nu_x) = 1$ . Additionally, we can bound the integral over the temperature dependent frequency error by

$$\min \left( \int_0^t g(\kappa(\tau)) d\tau \right) = t \cdot \delta_{\min} \quad (15)$$

$$\max \left( \int_0^t g(\kappa(\tau)) d\tau \right) = t \cdot \delta_{\max} \quad (16)$$

$$\min \left( \int_{-T}^0 g(\kappa(\tau)) d\tau \right) = T \cdot \delta_{\min} \quad (17)$$

$$\max \left( \int_{-T}^0 g(\kappa(\tau)) d\tau \right) = T \cdot \delta_{\max}, \quad (18)$$

where  $\delta_{\min}$  and  $\delta_{\max}$  are the minimum and maximum temperature dependent frequency error of the employed clock circuit. Putting it all together, we find that the maximum time synchronization error is bounded by

$$\varepsilon_Q(t = T) \leq 2f_0 T \cdot (1 + \delta_{\max}) - \frac{(f_0 T(1 + \delta_{\min}) - 1)^2}{f_0 T + 1} - (f_0 T - 1). \quad (19)$$

Figure 2 illustrates the worst-case synchronization accuracy for a given time interval between synchronization messages. We can see that there are two different error regions:

- (1) The synchronization error is dominated by quantization error;
- (2) The synchronization error is dominated by temperature induced frequency drift;

We can show that in the two regions, the maximum synchronization error approaches

$$\varepsilon_Q(T) \approx 4 \quad (20)$$

for the quantization dominated region, and

$$\varepsilon_Q(T) \approx f_0 T \cdot (2\delta_{\max} - 2\delta_{\min} - \delta_{\min}^2) \quad (21)$$

for the temperature induced frequency drift region. See Appendix A for a detailed derivation of these approximations.

To achieve an optimum between small time synchronization error and low message exchange rate, a time synchronization protocol has to operate exactly at the border of these two regions. We can find this point by equating Equation (20) with Equation (21) and solving for  $T$ . Thus, the optimum resynchronization rate assuming worst-case synchronization errors is

$$T^* = \frac{4}{f_0 \cdot (2\delta_{\max} - 2\delta_{\min} - \delta_{\min}^2)}. \quad (22)$$

**4.2.2 Introducing Time-Stamping Errors.** Timestamping error affects the precision with which a node estimates its current frequency error  $\delta_Q(t)$  and we can model it as

$$\delta_Q(0) = \frac{c(-T) - c(0) + \eta}{c_0(-T) - c(0) + \eta} - 1, \quad (23)$$

where  $\eta$  is the time-stamping error. Propagating this change, we find that the maximum synchronization error can be expressed as

$$\varepsilon_Q(t = T) \leq 2f_0 T \cdot (1 + \delta_{\max}) - \frac{(f_0 T(1 + \delta_{\min}) - 1 - \eta)}{f_0 T + 1 + \eta} \cdot (f_0 T(1 + \delta_{\min}) - 1) - (f_0 T - 1). \quad (24)$$

The effect of this change on the graph is a raised error in region (1), where quantization error dominates. However, over longer synchronization intervals, these time-stamping errors become negligible, and thus the approximation for the second region does not change.

### 4.3 A More Realistic Bound on the Time Synchronization Error

Our approximations made in Equations (16) to (18) are very conservative due to the worst-case modeling of temperature induced frequency error. It is highly unlikely that in a real system, a clock switches from the lowest possible frequency error, to the highest possible frequency error within two time synchronization intervals. For most embedded systems that are not used in extreme environments, like rockets or outer space, the temperature changes gradually and we can find a tighter bound for the time synchronization error by analyzing actual temperature traces.

Assume that we are given a temperature trace of the environment where our embedded system is located. We can scan this temperature trace and search for the biggest change in frequency error in a window of  $2T$ . Marking the minimum

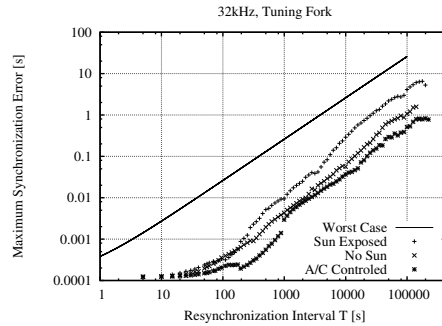


Fig. 3. Effects of different temperature environments. This graph shows the effect of using the maximum drift change calculated from a temperature trace on the maximum synchronization error. In general, the smaller the temperature changes, the better the synchronization accuracy for longer resynchronization intervals.

of this frequency error with  $\delta_{\min}$  and the maximum with  $\delta_{\max}$  we can reevaluate Equation (19) and get a more realistic bound on the maximum time synchronization error.

We collected three different temperature traces for three different environments. The three traces are each 7 days long, and have a time resolution of 5 seconds. We collected the data in a A/C controlled not sun exposed, a sun exposed not AC controlled, and a shaded not A/C controlled indoors area. Figure 3 shows the result for each of these temperature traces and the estimated maximum time synchronization error bound. We can conclude that the more variation there is in the change of temperature, the worse the expected time synchronization error bound becomes. However, we can still see a clear cut between the two regions where (1) the quantization error dominates and (2) where the temperature induced frequency drift dominates.

#### 4.4 Verification Through Simulation

There are many different sensor network simulators, each providing a different set of features and tools. However, none of the existing sensor network simulators captures the change in frequency error that occurs due to changes in local temperature. At best, some simulators model different static clock frequency offsets [Boulis 2007; Varshney et al. 2007], but these offsets are static over the time of a simulation, and are thus not useful for our experiments.

We enhanced Castalia [Boulis 2007] with a revised clock model, that takes as input a temperature trace and adapts the clock's frequency error at runtime. In addition, we added a realistic time-stamping mechanism that models the time-stamping errors found on the CC2420 radio chip used in many sensor network platforms. We implemented FTSP [Maróti et al. 2004] in Castalia to simulate a simple time synchronization protocol.

To verify our theoretic analysis, we run several simulations with changing resynchronization intervals. Figure 4 shows the 95% confidence intervals for the measured maximum time synchronization error, and compares the result from theory and

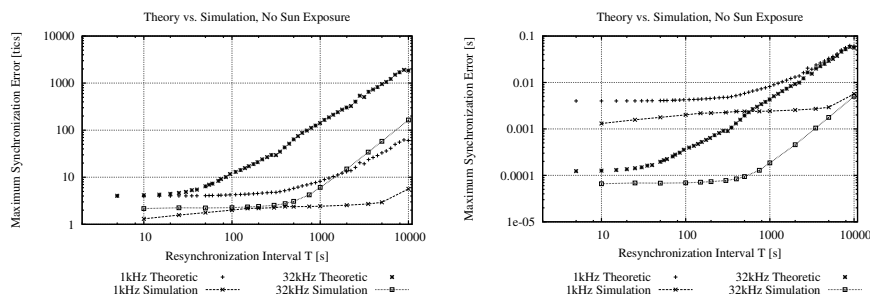


Fig. 4. This graph compares the theoretic upper bound to the 95% confidence interval found through simulating a time synchronization protocol. The simulated data shows that a synchronization protocol will be well below the theoretic upper bound. In both cases, theory and simulation, the limiting factor is the almost linear increase due to change in temperature. The higher frequency clocks hit this limit faster since they have a higher temporal resolution. Thus, for higher frequencies the divergence rises faster than for lower frequencies.

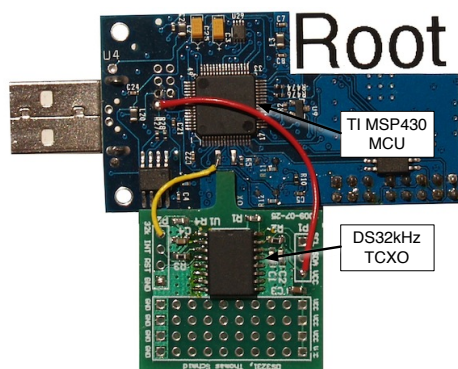


Fig. 5. Maxim DS32kHz Extension Board. This picture shows the modified basestation mote with an accurate TCXO timesource. While the DS32kHz can draw as little as  $6 \mu W$  in battery backup mode, it is almost the same size as the main MCU of the platform.

simulation for two different clock speeds. The simulations verify that our bounds on the temperature induced frequency drifts are indeed upper bounds, and that in simulation the time synchronization errors are much smaller than the bounds themselves.

#### 4.5 Verification on Hardware

We constructed a small network of seven TelosB motes, a popular sensor network platform consisting of a 8MHz Texas Instruments MSP430 16-bit microcontroller, a Texas Instruments CC2420 radio chip, and an external 32kHz crystal. The seven nodes were co-located in a sun-exposed environment such that they experienced approximately the same temperature changes. An additional TelosB mote acting as the base node was modified to contain a Maxim DS32kHz TCXO and used as

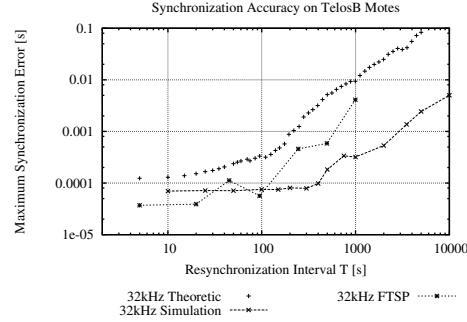


Fig. 6. Comparison of theoretic, simulation, and real hardware. A small network of seven nodes collected the synchronization accuracy data. A specially modified node with a 32 kHz TCXO was used as time reference.

time reference for the network. Figure 5 shows a picture of the extension board attached to the TelosB mote. The ground connection is provided by a large surface pad, while power to the TCXO is provided through a patch wire. A second wire connects the accurate 32 kHz clock signal of the TCXO to the clock input pin of the MCU. The other seven nodes ran FTSP in order to synchronize to that base node. The synchronization period of each node was set to a different time interval. This allows us to see the effect of different resynchronization intervals on the time synchronization accuracy.

We collected the synchronization accuracy from the seven nodes over 6 days in order to observe the effects of several diurnal cycles. Figure 6 compares the result from theory, simulation, and real hardware. We can observe that the data from the seven nodes is not as smooth as the data collected from simulation. This comes from the smaller number of data points available from real hardware. In simulation, we are able to have several nodes per synchronization interval, and we can repeat the simulation with different phase offsets and randomly chosen frequency drift curves. This is not possible on real hardware deployed in a real environment, since a temperature trace can not be replayed. Despite this limitation, we can see that the data from the seven nodes still retains the two-phase behavior that we observed in the theoretic and simulation data.

#### 4.6 Frequency Error Estimation

The frequency error estimation in a time synchronization algorithm is key to increased synchronization accuracy. Especially for longer synchronization intervals, the more accurate the frequency error estimation, the less error accumulates between the synchronization attempts.

Assume a reference node  $A$  sends out two beacons at time  $t_1$  and  $t_2$  respectively. Each beacon contains the precise time count  $c_A(t)$  of the reference node's clock. In this situation, a node  $B$  can estimate its current frequency error relative to node  $A$ 's clock using

$$\hat{\delta}_e(t_1, t_2) = \frac{(c_A(t_2) - c_B(t_2)) - (c_A(t_1) - c_B(t_1))}{c_A(t_2) - c_A(t_1)}. \quad (25)$$

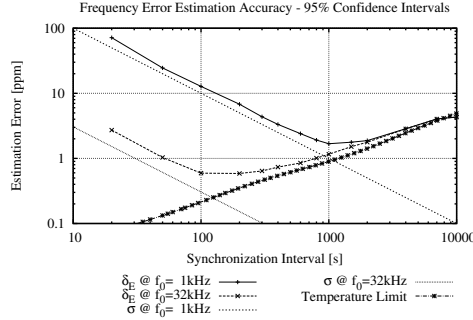


Fig. 7. Accuracy of frequency error estimation on a node. Two factors limit the accuracy of the frequency error estimation on a node: (1) quantization in for short resynchronization periods and (2) temperature at longer resynchronization periods.

Without loss of generality, assume that  $c_A(t_1) = c_B(t_1) = 0$  and  $t_2 = T$ . Thus, the frequency error estimation simplifies to

$$\hat{\delta}_e(T) = \frac{c_A(T) - c_B(T)}{c_A(T)}. \quad (26)$$

Equation (26) represents the average frequency error over the last resynchronization interval. Therefore, if we knew the actual frequency error of the node at time  $T$ , we could calculate the error of the frequency error estimation incurred at the node as

$$\delta_E(T) = \hat{\delta}_e(T) - g(\kappa(T)). \quad (27)$$

In hardware, it is difficult to obtain  $g(\kappa(t))$  since a frequency counter would have to be connected to the clock signal in parallel to the microcontroller. However, this value is readily available in our simulation and we can study the impact of temperature and quantization on the accuracy of the frequency error estimation with varying resynchronization intervals.

Figure 7 shows the 95% confidence intervals of the error in frequency error estimation for two different clock frequencies. We can observe, as in the case of synchronization accuracy, that there is an optimal resynchronization period at which the error confidence is minimized. If a node resynchronizes faster than the optimal point, then quantization will introduce error into our estimation. Since the system uses digital counters, the minimal resolution with which a node can estimate the frequency error is

$$\sigma = \frac{1}{f_0 \cdot T}. \quad (28)$$

At the same time, if a node resynchronizes slower than the optimal time, the assumption that the frequency error stays constant within a synchronization period gets violated because of change in temperature. Thus, temperature becomes the significant error source and the frequency error estimation loses accuracy.

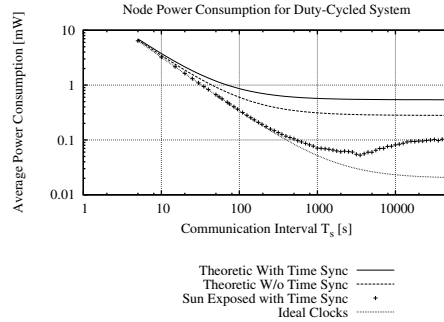


Fig. 8. This graphs shows the average power consumption for different synchronization techniques in a duty cycled system. With ideal clocks, no overhead is required, and thus a minimum power consumption is achieved. Using time synchronization, a duty-cycled system can greatly reduce its guard bands and thus save a lot of energy.

## 5. SYNCHRONIZATION, DUTY-CYCLING, AND POWER

We discussed the notion and need of duty cycling in Section 3.1, and we now have the tools to add to that analysis the impact of time synchronization on duty cycled systems. More precisely, we introduced the need for guard bands and defined them as

$$T_{gX} = 2 \cdot T_s \cdot s_X, \quad (29)$$

where  $T_s$  is the sleep time, and  $s_X$  the node's clock stability. Using the worst-case time synchronization error found from our simulation results in Section 4.4, we can redefine the guard band for a node with time synchronization as

$$T_g = 2 \cdot \varepsilon_Q. \quad (30)$$

This bound guarantees that a node that wakes up within  $T_g$  can communicate with its neighbors, even if it experiences the worst-case synchronization error.

Given the guard band of a duty cycled system without time synchronization and one with time synchronization, we can calculate their energy consumption using Equation (2). Figure 8 depicts this comparison using the same system parameters as was used in Section 3.2. Figure 8 includes the ideal case power consumption where the guard band  $T_g = 0$ .

It is not surprising that in the theoretic worst-case synchronization error calculations, a system without time synchronization fares better than one with time synchronization. The cause of this is that in the worst-case, the frequency error estimation for  $\delta_Q$  can introduce twice as much error as without time synchronization. However, in reality the occurrence of such an error is highly unlikely. Thus, using the more realistic time synchronization error bounds developed in Section 4.3 shows that using time synchronization can greatly improve the power efficiency of a duty cycled system.

This analysis ignores the power overhead that a time synchronization protocol introduces. Unfortunately, this overhead is poorly studied and we can only speculate on how much this overhead would be in a real system. In general, if a system

needs less than 10 ms accuracy, then a time synchronization interval of up to 1000 seconds is sufficient to guarantee such accuracies (see Section 4.4 and Figure 4). In most systems, this interval is much larger than the communication interval necessary to transmit sensor data from the nodes to a fusion center and we can assume that the overhead of time synchronization in these scenarios is minimal. However, if high accuracy is needed, the synchronization intervals have to be of the order of tens of seconds. In these cases, time synchronization could be piggy-backed on regular data messages, only introducing a small message overhead of the order of a timestamp.

## 6. CONCLUSION

The time synchronization service in sensor networks is tightly connected to the radio, clocking hardware, and synchronization algorithm employed. Naturally, improvements can be made on all levels of the system. Over the last years, most improvements were made in the synchronization algorithm trying to minimize message exchange and optimize radio architectures to provide accurate time-stamping mechanisms. The influences of the underlying clock system and its impact on the overall synchronization accuracy has largely been unstudied.

Our first key contribution illustrates that choosing the right clock for a sensor network platform is an important factor for the overall system performance. However, equally important is the environment in which a node resides. If a system is exposed to high temperature variations, a higher resynchronization rate is necessary to cope with the change in frequency. Our analysis showed how temperature affects the accuracy of a synchronization protocol, and gives an idea of what a system architect has to expect if he tries to optimize the power consumption of a synchronization protocol by enlarging the resynchronization period.

As our second contribution, we showed that the error in frequency error estimation during short synchronization periods can be substantial, and in extreme cases even decrease the overall synchronization accuracy and unnecessarily increase the energy consumption. For example, with a low frequency clock of 1 kHz, even at a resynchronization period of 100 second the overall confidence interval is still above 10ppm. In these scenarios, performing frequency error estimation is a waste of energy since it won't increase the synchronization accuracy at all.

Our last contribution is an overall insight into the relationship of the power consumption in duty-cycled systems. We illustrate how time synchronization can significantly reduce the guard bands necessary because of clock inaccuracies. Similar effects can be achieved by using more stable hardware clocks. But a careful evaluation of their energy consumption has to be performed in order to achieve an overall reduction. Often, the gains made by reducing guard bands can easily be offset by the increased power consumption of a more stable clock.

## APPENDIX

## A. SYNCHRONIZATION ERROR APPROXIMATION

In Section 4.2.1 we found that the synchronization error is bounded by

$$\varepsilon_Q(T) \leq 2f_0T \cdot (1 + \delta_{\max}) - \frac{(f_0T(1 + \delta_{\min}) - 1)^2}{f_0T + 1} - (f_0T - 1). \quad (31)$$

In region one, we assume that the frequency error  $\delta = 0$ , and thus the synchronization error simplifies to

$$\varepsilon_Q(T) \leq 2f_0T - \frac{(f_0T - 1)^2}{f_0T + 1} - (f_0T - 1). \quad (32)$$

We can now calculate the limit and find

$$\lim_{T \rightarrow \infty} \varepsilon_Q(T) = 4. \quad (33)$$

In the second region, where the frequency drift overshadows the quantization errors, the synchronization error reduces to

$$\begin{aligned} \varepsilon_Q(T) &\leq 2f_0T \cdot (1 + \delta_{\max}) - \frac{(f_0T(1 + \delta_{\min}))^2}{f_0T} - f_0T \\ &= 2f_0T \cdot (1 + \delta_{\max}) - f_0T \cdot ((1 + \delta_{\min}))^2 - f_0T \\ &= f_0T \cdot (2 + 2\delta_{\max} - 1 - (1 + \delta_{\min})^2) \\ &= f_0T \cdot (2\delta_{\max} - 2\delta_{\min} - \delta_{\min}^2) \end{aligned} \quad (34)$$

## ACKNOWLEDGMENTS

This material is supported in part by the U.S. ARL and the U.K. MOD under Agreement Number W911NF-06-3-0001, and by UCLA's NSF Science and Technology Center for Embedded Networked Sensing. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the listed funding agencies. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

## REFERENCES

- AEBISCHER, D., OQUEY, H., AND VON KAENEL, V. 1997. A 2. 1-MHz crystal oscillator time base with a current consumption under 500 nA. *IEEE Journal of Solid-State Circuits* 32, 7, 999–1005.
- BOULIS, A. 2007. Castalia: revealing pitfalls in designing distributed algorithms in wsn. In *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, Sydney, Australia, 407–408.
- COOKLEV, T., EIDSON, J. C., AND PAKDAMAN, A. 2007. An implementation of ieee 1588 over ieee 802.11b for synchronization of wireless local area network nodes. *IEEE Transactions on Instrumentation and Measurement* 56, 5, 1632–1639.
- DUTTA, P., CULLER, D., AND SHENKER, S. 2007. Procrastination might lead to a longer and more useful life. In *Proceedings of the Sixth Workshop on Hot Topics in Networks (HotNets-VI)*. ACM, Calgary, Alberta, Canada.
- DUTTA, P., TANEJA, J., JEONG, J., JIANG, X., AND CULLER, D. 2008. A building block approach to sensor network systems. In *Proceedings of the Sixth ACM Conference on Embedded Networked Sensor Systems (SenSys)*. ACM, Raleigh, North Carolina, USA, 267–280.
- EIDSON, J. C. 2006. *Measurement, Control, and Communication Using IEEE 1588*. Springer, New York, NY, USA.
- ELSON, J., GIROD, L., AND ESTRIN, D. 2002. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the 5th symposium on Operating systems design and implementation (OSDI)*. Vol. 36. ACM, New York, NY, USA, 147–163.
- GANERIWAL, S., GANESAN, D., SHIM, H., TSIATSIS, V., AND SRIVASTAVA, M. B. 2005. Estimating clock uncertainty for efficient duty-cycling in sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems (SenSys)*. ACM Press, New York, NY, USA, 130–141.
- HAQUE, M. AND COX, E. 2004. *Use of the CMOS Unbuffered Inverter in Oscillator Circuits*. Texas Instruments. Application Report.
- MARÓTI, M., KUSY, B., SIMON, G., AND LÉDECZI, Á. 2004. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*. ACM Press, New York, NY, USA, 39–49.
- MILLS, D. L. 1991. Internet time synchronization: the network time protocol. *IEEE Transactions on Communications* 39, 10, 1482–1493.
- SALLAI, J., KUSY, B., LEDECZI, A., AND DUTTA, P. 2006. On the scalability of routing integrated time synchronization. In *Third European Workshop on Wireless Sensor Networks (EWSN)*. iSpringer, Zurich, Switzerland.
- SCHMID, T., SHEA, R., FRIEDMAN, Z. M. C. J., SRIVASTAVA, M. B., AND CHO, Y. H. 2009. On the interaction of clocks and power in embedded sensor nodes. Tech. rep., University of California, Los Angeles, Networked and Embedded Systems Lab.
- SIMON, G., MARÓTI, M., LÉDECZI, Á., BALOGH, G., KUSY, B., NÁDAS, A., PAP, G., SALLAI, J., AND FRAMPTON, K. 2004. Sensor network-based countersniper system. In *Proceedings of the 2nd international conference on Embedded networked sensor systems (SenSys)*. ACM Press, New York, NY, USA, 1–12.
- SUNDARARAMAN, B., BUY, U., AND KSHEMKALYANI, A. 2005. Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks* 3, 3, 281–323.
- VARSHNEY, M., XU, D., SRIVASTAVA, M., AND BAGRODIA, R. 2007. sQualNet: A scalable simulation and emulation environment for sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems (IPSN)*. ACM Press, New York, NY, USA.