High-Resolution, Low-Power Time Synchronization an Oxymoron No More

Thomas Schmid†, Prabal Dutta‡, Mani B. Srivastava†

†Electrical Engineering Department
University of California, Los Angeles
Los Angeles, CA 90095
{thomas.schmid, mbs}@ucla.edu

‡Computer Science & Engineering Division
University of Michigan
Ann Arbor, MI 48109
prabal@eecs.umich.edu

ABSTRACT

We present Virtual High-resolution Time (VHT), a power-proportional time-keeping service that offers a baseline power draw of a low-speed clock (e.g. 32 kHz crystal), but provides the time resolution that only a higher frequency clock could offer (e.g. 8 MHz crystal), and scales essentially linearly with access (i.e. the "reading" and "writing" of the clock). We achieve this performance by revisiting a basic assumption in the design of time-keeping systems – that to achieve a given time-stamping resolution, a free-running timebase of equivalent frequency is needed. We show that this assumption is false and argue that the dependence is not on usage (i.e. whether on or off) but rather on access (i.e. reading and writing). Therefore, it is possible to duty cycle the free-running timebase itself, and augment it with a lower-frequency, temperature-compensated one, which achieves comparable resolution, at a fraction of the power, for typical workloads. The key technical challenge lies in duty cycling the fast clock and synchronizing the fast and slow clocks. To assess the viability of the approach, we explore how VHT could be implemented on several different platform architectures, and to study the power/performance tradeoff, we characterize VHT on one particular architecture in detail. Our results show power-proportional operation with a 10× improvement in average power and a synchronization accuracy exceeding 1 μs at duty cycles below 0.1%.

Categories and Subject Descriptors

B.m [**Hardware**]: Miscellaneous

General Terms

Experimentation, Measurement

Keywords

Clocks, Time Synchronization, Virtual High-resolution Time, Low-power

1. INTRODUCTION

Time synchronization is an important service in sensornets and many resources have been devoted to improving its accuracy and precision. However, the overall mean synchronization accuracy has not improved appreciably below about 1.5 μs, and *low-power* time synchronization at that level of accuracy has been altogether absent. The reason for this absence is two-fold. First, radio hardware support is needed to achieve high-precision message *time-stamping*,[1] which may or may not be feasible on a particular platform. Second, high-frequency clocks are needed to achieve high-precision *time-keeping*,[2] but such clocks simply draw too much power relative to the sleep mode power of duty-cycled sensor nodes, and are therefore rarely used in low-power settings.

Conventional wisdom holds that to achieve a certain level of time synchronization precision, a free-running timebase of comparable frequency is needed. Since system power draw is given by $P \propto CV^2 f$, where P is power, C is the switching capacitance, V is the operating voltage, and f is the switching frequency, holding all other things constant, it would appear that time-keeping costs grow essentially linearly with the required precision. We find that for a typical sensor node, the difference in power draw from a 32 kHz crystal and an 8 MHz crystal is a factor of 340, which is reasonably close to the expected factor of 244. This observation would appear to confirm the conventional wisdom that low-power and high-precision are incompatible, but we argue otherwise.

In this paper, we claim that high-resolution, low-power time synchronization is an oxymoron no more. First, a growing number of modern, low-power radios already support sub-microsecond level message time-stamping with sufficiently high accuracy and precision, addressing the first challenge. Second, we show that by combining a pair of clocks – one low-frequency (e.g. 32 kHz) and one high-frequency (e.g. 8 MHz) – and by powering the high-frequency clock only when timestamps are to be taken or when precisely time-triggered events must occur, we can achieve both low-power and high-precision, and offer power-proportional operation that scales essentially linearly with access (i.e. reading or writing the clock) rather than with operation (i.e. the clock is on or off). This distinction is subtle but important because it enables the fast clock to be duty-cycled so that it is powered down between accesses.

[1]Message time-stamping is the act of associating a specific time to the transmission or reception of a message.

[2]Time-keeping is the process of maintaining an accurate representation of time between resynchronization attempts.

The basic idea behind our approach, called Virtual High-resolution Time (VHT), is to use the high-frequency clock to *improve precision on-demand* but otherwise leave it turned off. The key insight is that one can achieve high-precision by using a low-frequency clock to obtain coarse-grained time, and use a high-frequency clock to provide fine-grained precision *relative to the coarse-grain timebase.* If the coarse and fine-grained clocks can be synchronized, perhaps using a phase-locked loop (PLL) as is common in analog and digital VLSI designs, then all time ticks are phase-aligned. Even if the two clocks run independently, the relative phase error (or jitter) with respect to a phase-locked approach is limited to $\pm T_f/2$, where T_f is the period of the fast clock.

We show how our approach can be implemented using an off-the-shelf microcontroller or a field-programmable gate array. The former allows the VHT design to be put into practice immediately while the latter outlines a roadmap for integrating the ideas into future silicon. We also study VHT's basic power/performance tradeoff and characterize VHT in depth on the Epic and TMote Sky platforms. Our empirical results show power-proportional operation with a $10\times$ improvement in average power and a synchronization accuracy exceeding 1 μs at duty cycles that are effectively below 0.1%. The key result is that the sleep mode power of high-resolution time synchronization no longer dominates the system power budget, opening up new applications.

As battery-operated and power-constrained nodes begin to keep time with high-precision and low-power, applications like beam-forming [1], compressive sensing [2], and improved time-of-flight ranging [3] will emerge or become more popular. And as nodes achieve accurate, precise, and low-power time-keeping, the benefits will accrue to node lifetime as well, since time drift-induced idle listening guard bands can be cut short [4]. New industrial applications will also emerge. The IEEE 1588 Precision Timing Protocol, for example, was developed because NTP, the *de facto* Internet time synchronization standard, was not precise enough for industrial measurement and control systems. However, IEEE 1588 requires sub-microsecond accuracies, a level of precision not attained by prior work [5, 6], but well within the reach of current technology at a power draw that is digestible to many low-power wireless sensor network applications.

2. ACHIEVING HIGH PRECISION

The two key primitives needed to achieve ϵ-precision time synchronization are

1. a high-resolution clock source with frequency f_0, and
2. message time-stamping with accuracy $\epsilon < \pm 1/f_0$.

If a wireless node can provide both of these primitives, then achieving precise time synchronization becomes a trivial matter of measuring clock offsets between nodes in the network and applying one of the many existing frequency error estimation techniques.

2.1 Importance of High-frequency Clocks

It should be clear that the higher the clock frequency, the finer the time resolution with which message reception or transmission can be timestamped. This is essential for high-precision time synchronization, since a clock can only estimate an event time to a resolution of $1/f_0$.

Another important aspect of high-precision time synchronization is the accuracy with which the node can actually estimate its current frequency error with respect to a reference clock. Without this error correction, nodes would have to resynchronize at relatively short intervals. Frequency error estimation can be accomplished easily using two time offset measurements as follows. Assume that a reference node A sends two time beacons, the first one at time t_1 and the second one at time t_2. Each beacon contains the quantized time count $c_A(t)$ observed at the reference clock when the particular beacon was transmitted. A node B can estimate its current frequency error relative to node A's clock using its local clock counter as

$$\hat{f}_e = \frac{(c_A(t_2) - c_B(t_2)) - (c_A(t_1) - c_B(t_1))}{c_A(t_2) - c_A(t_1)}, \tag{1}$$

where $c_A(t) = \lfloor f_0 \cdot t \rfloor$ is the reference clock's accurate time count, and $c_B(t) = \lfloor (f_0 + f_e) \cdot t \rfloor$ is the value of the counter at node B. Without loss of generality, let us assume that $c_A(t_1) = c_B(t_1) = 0$, i.e. that both of the nodes started counting from zero, and define $T = t_2$. Equation 1 becomes

$$\hat{f}_e = \frac{c_A(T) - c_B(T)}{c_A(T)} \tag{2}$$

From this relationship, we can determine the limit to the resolution with which the frequency error can be estimated. Since the counters on both nodes A and B are quantized digital systems, and the counters can increment only in unity steps, we calculate the minimum resolution of the frequency error estimation by taking the difference between two measurements that are separated by one tick, i.e.,

$$\begin{aligned}\delta_Q &= \frac{c_A(T) - c_B(T)}{c_A(T)} - \frac{c_A(T) - (c_B(T) + 1)}{c_A(T)} \\ &= \frac{1}{c_A(T)} \\ &= \frac{1}{T \cdot f_0}.\end{aligned} \tag{3}$$

To make the analysis more concrete, for an $f_0 = 8$ MHz clock, if the two beacons are $T = 10$ s apart, then the frequency error resolution is 0.0125 ppm, while for a low-frequency clock where $f_0 = 32.768$ kHz, the resolution decreases to 3.05 ppm. Thus, high frequency clocks are needed not just for time-stamping the reception or transmission of a message with a high time resolution, but they are also needed to improve the frequency error estimation over intervals where the change in frequency error due to changes in environmental temperature can be assumed constant. Or, put another way, the greater the clock frequency, the shorter the interval of time needed to synchronize a pair of clocks to a given frequency error resolution.

2.2 Importance of Accurate Timestamps

Message time-stamping is the operation of associating a time with a received or transmitted message. If this operation is either inaccurate or imprecise, then overall time synchronization accuracy will suffer. It has been shown that time-stamping in software above the MAC layer is problematic since software interrupts and non-deterministic jitter often make obtaining accurate timestamps difficult or impossible. Thus, a range of techniques have been developed for time-stamping, often targeted to the architecture and features of a particular radio chip.

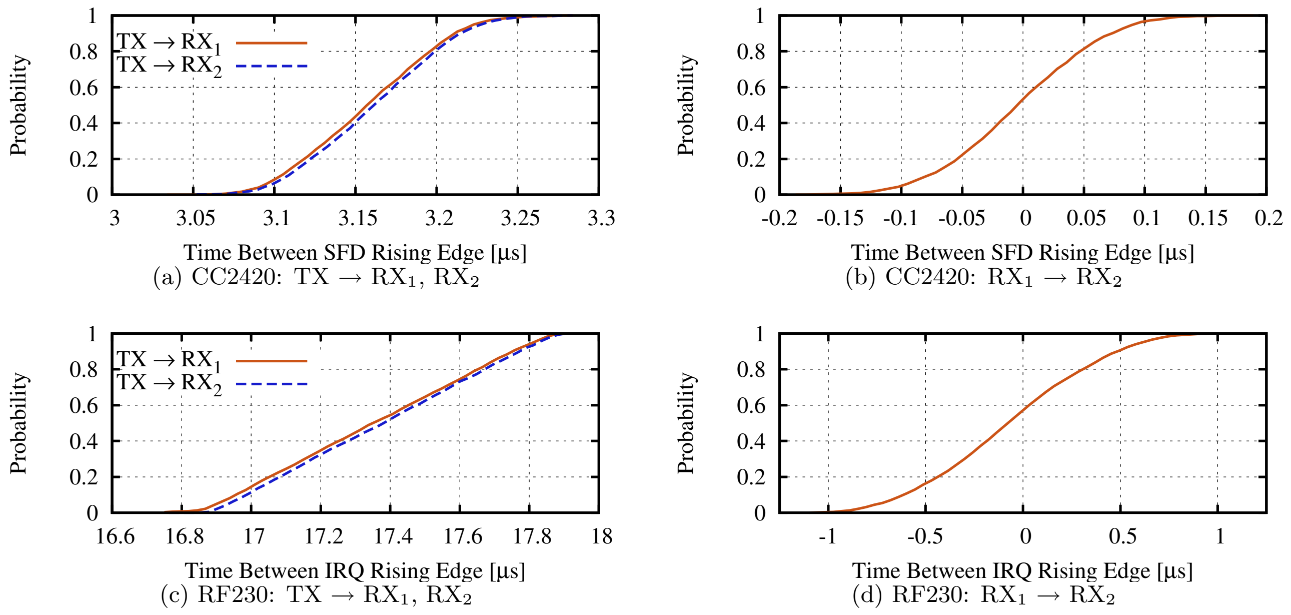

Figure 2: Cumulative distribution of the empirical time difference between the rising edge of the interrupt lines on three TI CC2420 and three Atmel RF230 radios. Figure 1 shows the experimental setup. The CC2420 exhibits approximately five times better time-stamping precision than the RF230.

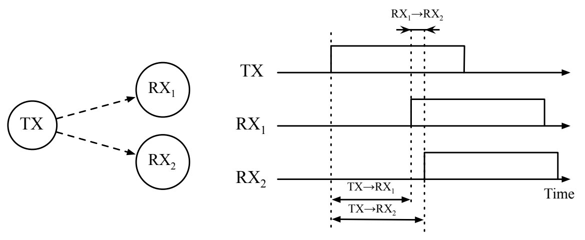

Figure 1: Experimental setup for the SFD/interrupt timing measurements.

2.2.1 *Message Time-stamping in Sensor Networks*

Most modern radios export an interrupt line that fires at a particular point during the reception or transmission of a radio message. This interrupt is commonly connected to a microcontroller capture line. One of the earliest attempts at time synchronization in sensor networks [7,8] used the interrupt of the first byte sent or received by the TI CC1000 [9] radio. While this worked reasonably well, and accuracies of approximiately 10 μs were achieved using clock speeds of $\sim$1 MHz, a more robust technique was developed for use with the Flooding Time Synchronization Protocol (FTSP) [5]. Instead of time-stamping just the first received or transmitted byte, FTSP timestamped several bytes of a message. By profiling the CC1000 radio chip, the FTSP developers determined the precise timing with which these interrupts arrive, allowing them to infer a more accurate overall message timestamp. This particular technique allowed FTSP to achieve a one-hop mean accuracy of 1.5 μs, which was at the time unprecedented in wireless sensornets.

Since then, FTSP has been the *de facto* standard for time synchronization, and only recently have improvements emerged for superior multi-hop performance [10]. However, even the newer work uses the same time-stamping mechanism that was originally developed for FTSP years ago.

2.2.2 *Message Time-stamping on 802.15.4 Radios*

The Texas Instruments CC2420 and Atmel AT86RF230 are two common IEEE 802.15.4-compliant radios used in popular sensor network platforms. The CC2420 was first introduced to the sensor network community on the Telos platform, where it was combined with a Texas Instruments MSP430 microcontroller, and the MicaZ platform with an Atmel ATMega128L microcontroller, while the RF230 is the radio used on the Crossbow Iris platform. While the IEEE 802.15.4 standard comprises three different frequency bands, the CC2420 and RF230 use the 2.4 GHz band. This means they use an O-QPSK modulation and demodulation at a 2 MChip/s rate, resulting in an effective channel bandwidth of 1 MHz. Using a spreading factor of eight for process gain, the maximum data-rate is 250 kbit/s.

An important part of the IEEE 802.15.4 standard is the definition of the physical and link layer frame format. Each frame consists of a 4 byte preamble, 1 byte Start of Frame Delimiter (SFD), 1 byte Frame Length, 2 byte Frame Control Field (FCF), 1 byte sequence number, 0 to 20 bytes address information, 2 bytes of Frame Check Sequence, and n bytes of frame payload. This frame structure is important for understanding of the event time-stamping mechanism of the CC2420 radio, while the RF230 makes only tangential use of it. The CC2420 has a dedicated interrupt line that indicates the moment when the SFD field has either been sent by the radio (in case of frame transmission), or received by the radio (in case of a frame reception).

In the case of the RF230, a single interrupt line is multiplexed among a range of different radio state machine events. This multiplexing makes it much more complicated to use for message time-stamping since some of the state machine events depend on exogenous factors like the length of the received message.

Since the timing of an interrupt on this line can be captured using a hardware timer unit, precise time-stamping is available on the CC2420 radio. But the question of just how accurate and precise this mechanism is on the CC2420, and how well it compares to the RF230, remain unexplored in the literature to the best of our knowledge. Figure 1 illustrates an experimental setup we use to explore this question in detail.

Figures 2(a) and 2(b) illustrate a sequence of measurements performed on three sensor network nodes using the CC2420 to characterize the achievable time-stamping precision of the radio. In these experiments, one node transmits and two nodes concurrently received a frame. We measure the time elapsed between the SFD interrupt line rising at the transmitter and the SFD line rising at the two receivers. We observe that the distribution of measurements is almost identical from the transmitter to receiver 1, and transmitter to receiver 2, with a mean of 3.162 μs and 3.166 μs respectively, and a standard deviation of 41.26 ns and 40.90 ns. Figure 2(a) shows that 95% of the measurements fall within an interval of only 160 ns.

We next explore the distribution of the difference in SFD interrupt times at the two receivers. If the time differences were negligible, then receiver-receiver synchronization, as employed by the RBS scheme [7], would achieve high pairwise synchronization since nearly all non-determinism would be eliminated. Unfortunately, as Figure 2(b) illustrates, this is not the case. The figure shows the cumulative distribution of the time difference between the SFD rising edges at the two receivers. The distribution is essentially normal, with mean 3.58 ns and standard deviation 58.14 ns.

Given the need for an accurate message time-stamping primitive, we must employ a clock of frequency

$$f_0 \leq \frac{1}{\epsilon/2} = \frac{1}{80 \text{ ns}} = 12.5 \text{ MHz}$$

to guarantee with high probability that the message time-stamp can be resolved to within ± 1 clock tick. If this condition is satisfied, trivial equations involving only two offset measurements can still achieve high accuracy synchronization, without requiring costly regression algorithms.

We perform similar measurements using the RF230 radio, but instead of using the SFD interrupt, we use the TX done and RX done interrupt events. While these interrupts are not as precises as the SFD interrupt of the CC2420, since they depend on the message length, Figure 2(c) and 2(d) illustrate that they still have the potential to support high-precision time synchronization. The mean time between transmitter to receiver 1, and transmitter to receiver 2 are both 17.4 μs with a standard deviation of 290 ns and 370 ns respectively. While we can compensate for the mean error, the standard deviation is approximately 7$\times$ higher than the CC2420. Thus, the CC2420 will achieve an overall better time synchronization accuracy if high-frequency clocks are employed. However, at frequencies below 1 MHz, this distinction will disappear.

3. ACHIEVING LOW POWER

The two basic requirements for achiveing low-power time synchronization are

1. low-frequency clocks, and
2. infrequent communication.

The low-frequency clocks are necessary to minimize the power dissipation of the oscillator, digital counter, and clocking network as $P \propto CV^2 f$. This is a well-known fact in low-power embedded systems design, where 32 kHz[3] crystals are the norm for real-time clocks (RTC) and nearly all embedded systems that employ duty-cycled operation use such a subsystem for scheduling wakeup alarms. The power draw of a 32 kHz RTC ranges from a μA or so to several tens or hundreds of μA, depending on the configuration, load, and clock stability. For example, the entire Telos platform [11] draws only 7.2 μA at 3.0 V or 4.5 μA at 2.3 V[4] when the microcontroller is in sleep and a 32 kHz external crystal oscillator is clocking one of the timers.

But a low-frequency crystal is not the only thing necessary for low-power time synchronization. Communications overhead can dominate the power dissipation of the clock subsystem and hinder efforts at low-power operation. Thus, infrequent communication events are also necessary, which translates into large resynchronization intervals. However, the longer the resynchronization interval, the higher the probability that changing environmental temperatures – even small ones – will cause greater frequency error and accumulated drift between resynchronization attempts, and thus introduce an additional time synchronization error.

3.1 Achieving High Clock Stability

A hardware solution to the clock skew problem is the temperature-compensated crystal oscillators (TCXO). Many TCXOs are available, and they come in many forms ranging from low to high frequency, and some TCXOs also integrate a full RTC into a single package. While the power draw of a TCXO during temperature compensation is high, as Figure 3 illustrates, TCXO manufacturers recognize that temperature measurement and compensation can be done relatively infrequently since temperature is a relatively slow varying process in many environments. Thus, the duty-cycling of the temperature measurement circuitry dramatically lowers the average power draw of these devices, making their power draw viable for many sensor network applications. For example, the Maxim DS3231, a 32 kHz TCXO with an integrated RTC, draws an average of 19.82 μA at 2.3 V when it is operated in its battery backup mode while the DS32kHz draws even less power – just 2.3 μA at 2.3 V. This draw is half of what a Telos mote draws in deep sleep mode, without integrating any timer mechanism. The main advantage in using a TCXO is that the frequency stability increases from a typical -120 ppm to 10 ppm[5] to $\pm$2 ppm.

[3]The actual frequency is usually 32.768 kHz for easier division in binary systems.

[4]The platform's leakage current is 4.2 μA at 3 V and 2.4 μA at 2.3 V.

[5]32 kHz crystals are usually tuning fork crystals which have a quadratic frequency error vs. temperature curve. -120 ppm corresponds to the error at the low and high end of the temperature, while 10 ppm is normally achieved at about room temperature.

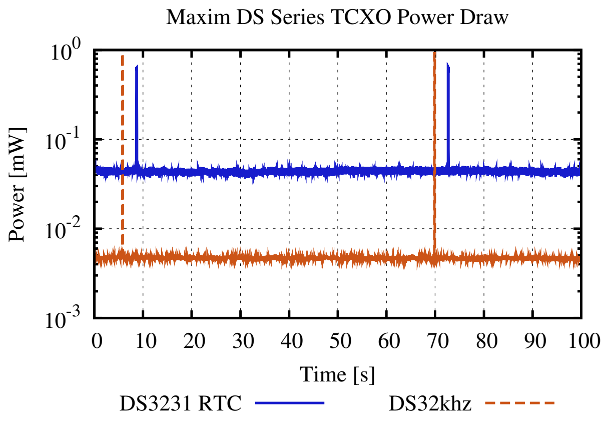

Figure 3: Power draw of two Maxim DS series low-power 32 kHz TCXO (DS3231 and DS32kHz). On both devices, the power was measured in battery mode and with a 100 kΩ pull-up resistor on the 32 kHz line for the DS3231.

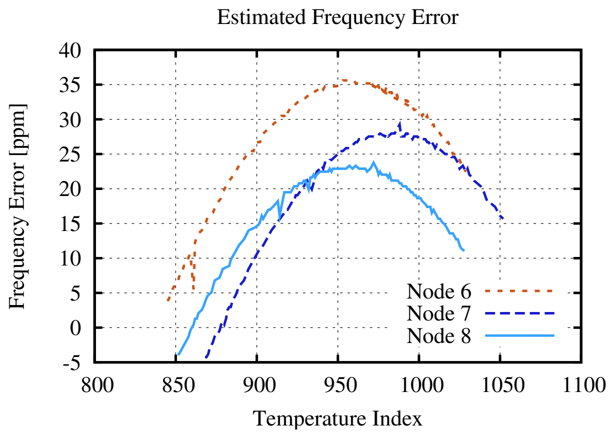

Figure 4: TCTS auto-calibrated $\hat{f}_e$ vs. temperature curve for three TMote Sky nodes. This figure illustrates the auto-calibration possibilities of TCTS.

While the energy consumption of TCXOs is small, the devices are rarely found on sensor network platforms because of their relatively large size and high cost compared to other components. For example, the DS3231 is approximately 10x10 mm, almost twice the area of the TI CC2420 radio, 1.2× the area of an MSP430F1611 microcontroller, and 4× the size of the more modern TI CC2520 radio. Since the trend in sensor network platforms is smaller size and lower cost, today's off-the-shelf TCXOs are not ideally suited to providing a high-stability, low-frequency clock source.

3.2 Temperature Compensated Time Sync

One alternative to TCXOs is temperature-compensated time synchronization (TCTS) [12]. The TCTS algorithm runs on a sensor network platform that is equipped with a regular temperature sensor. The algorithm dynamically selects an appropriate synchronization interval depending on the the observed temperature excursions. The main idea behind TCTS is as follows. Within a network of nodes, there is at least one reference node that has access to an accurate and stable timebase. This timebase could be a TCXO or perhaps even a GPS receiver that provides accurate time to the reference node. All other nodes in the network synchronize with the reference node(s) using a time synchronization protocol. The only requirement on that time synchronization protocol is that it can calculate the current frequency error with respect to the reference node's clock. Unlike other time synchronization protocols, TCTS also records the current temperature during a synchronization exchange. Both the temperature and frequency error at the end of the synchronization exchange is cached in a frequency vs. temperature table in memory.

Before attempting subsequent resynchronization, TCTS will measure the current temperature and consults its internal calibration table. If the current frequency error for the measured temperature is cached, TCTS will not attempt to resynchronize with the reference node since a new time estimate is not required. Eventually, when all of the operating temperatures have been observed, TCTS will have auto-calibrated the low-stability clock, essentially providing a TCXO timebase. The main advantage of this approach is that no additional hardware is needed, since temperature sensors are available on most any sensor network platform, thus removing the need for a separate TCXO, and the additional area and cost that it would demand, from the design.

3.2.1 Running TCTS on the TMote Sky

In order to show that TCTS works not just in simulation (as in prior work) but also in practice, we implemented the algorithm on the TMote Sky platform. The TMote Sky includes a 32 kHz external crystal oscillator, a CC2420 radio, and an MSP430F1611 microcontroller. This microcontroller, like many other embedded processors, contains an integrated, on-chip temperature sensor. While not ideal, since it measures the CPU die temperature instead of the crystal temperature, it is sufficient for our purposes, allowing TCTS to work reliably and calibrate for changes in environmental temperatures. However, we expect that TCTS would work even better if the temperature sensor were located close to the crystal itself.

For the evaluation of the TCTS algorithm, we chose a synchronization interval during calibration of $T = 100$ s. As other studies have shown [12], the 32 kHz crystal achieves the best frequency estimation accuracy using this interval, and using Equation (3), we calculate the minimum frequency estimation resolution for this setting as 0.3 ppm. The reference node is a modified TMote Sky in which the 32 kHz crystal has been replaced with a DS3231 TCXO.

Initially, we subject the nodes over several hours to a wide range of temperatures from -10 °C to 60 °C using a temperature chamber. The temperature is changed slowly in order to allow TCTS time to calibrate itself to each temperature setpoint. Figure 4 shows the extracted frequency error vs. temperature index curve for three different nodes. The temperature index is in terms of ADC readings since there is no need to convert it to standard temperature units (e.g. °C). As expected, the frequency error has a quadratic shape corresponding to a tuning-fork crystal oscillator, and the frequency error over the observed temperature range is 35 ppm. While the calibration process is not ideal, we note that un-

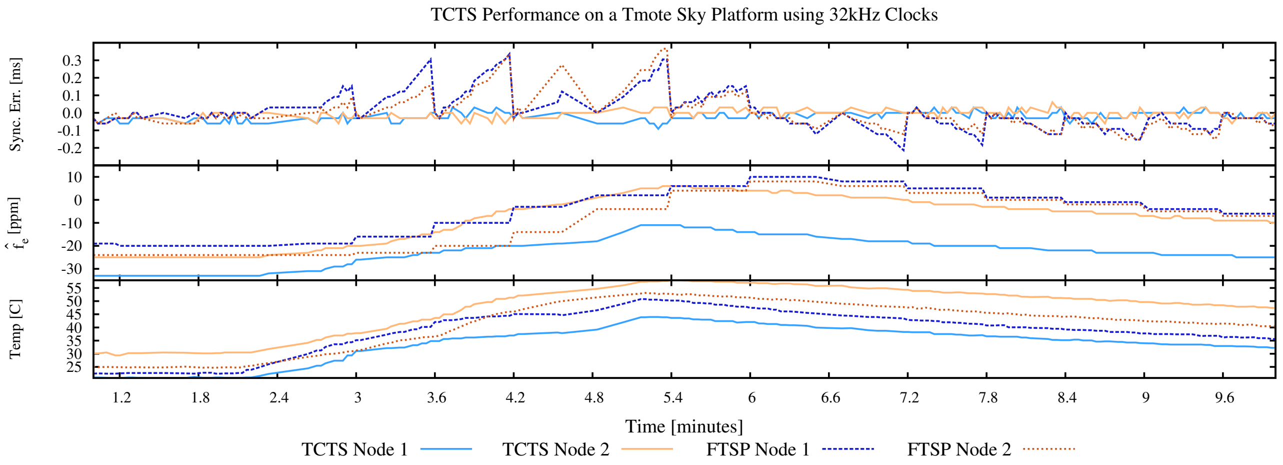

Figure 5: TCTS compensation accuracy. We subject four TMote Sky nodes, two running a TCTS enhanced version of FTSP and two running regular FTSP, to a high change in temperature, similar to a shade→sunshine transition. The x-axis tick marks indicate the 36 s resynchronization intervals. We observe how the TCTS nodes adapt their frequency error estimate $\hat{f}_e$ based on the measured temperature, while the FTSP nodes must wait until the synchronization event.

der actual, slow-changing environmental temperature conditions, the TCTS algorithm could take multiple temperature vs. frequency error measurements and average over time to improve and smooth the calibration curve. In addition, since we know that the curve should fit a quadratic function of the form

$$f_e(T) = -A \cdot (T - T_0)^2 + B,$$

where, A is the temperature coefficient, $T_0 = 20°\text{C}$ represents room temperature, and B is a frequency error offset, we can fit the measured calibration points to this curve and obtain frequency error estimates for previously unobserved temperatures, thus drastically improving the range and accuracy while eliminating a factory calibration step and allowing on-demand calibration.

To demonstrate the efficiency of TCTS, we subject four nodes, two running an unmodified version of FTSP and two running FTSP enhanced with TCTS, to a rapidly-varying temperature environment. The FTSP resynchronization rate is set to 30 seconds, but it could be much higher in a real deployment. This experiment illustrates how FTSP can desynchronize and calculate incorrent frequency error estimates $\hat{f}_e$ that it uses for compensation, while TCTS-enhanced nodes rely on their temperature sensor and calibration table to retrieve the new frequency error estimate. Figure 5 illustrates the result of the experiment. We see that although the frequency error estimate starts to change when the temperature rises for both node pairs, only the TCTS-enhanced nodes are able to cope with the change and maintain tight synchronization. Even more interesting is the effect on the node when the temperature begins to fall. While the nodes running TCTS are not affected, the offset measurements of the regular FTSP nodes get corrupted with wrong values, and thus the node drastically starts to undershoot the actual synchronization time. It takes several minutes for the nodes to recover.

While measuring temperature is not free – the example of the TCXO power measurements show that while they consume high power – it can be accomplished in a very short time. In our example hardware that uses the on-chip temperature sensor, a measurement takes 35 ms and draws an average of 600 μA, or a total of 66.5 μJ. This is an order of magnitude smaller then just sending one radio message over the wireless channel, which consumes about 600 μJ alone [13], disregarding that at least one other node has to spend roughly the same energy to receive that message.

4. VIRTUAL HIGH-RESOLUTION TIME

Section 2 concludes with the simple observation that high-frequency clocks and an accurate time-stamping mechanism are needed to achieve high accuracy. Section 3 advocates low-frequency clocks and infrequent synchronization for low-power operation. These two requirements stand in direct contradiction in conventional designs. Our key insight is that a high-frequency clock is needed only during the time-stamping operation while a low-frequency clock is needed during periods of system inactivity, and especially miccrocontroller and radio sleep times. This observation suggests a new design in which a high frequency clock is active only when the radio is active, and off when the radio is off (during which time the design relies on a stable, low-frequency clock). The key design question is how to construct such a system that can choreograph and synchronize the two clocks – one fast and one slow – to each other and expose a unified clock abstraction to system and application software. The remainder of this section describes a new timer system, Virtual High-resolution Time (VHT), and presents the concepts and methodology for achieving this coordination, and details on implementing it in a microcontroller or FPGA.

The basic idea behind VHT is that during active periods, the high-frequency clock is turned on, and a hardware counter counts the number of high-frequency clock ticks that occur during each low-frequency clock interval, i.e., there are

$$\varphi_0 = \frac{f_H}{f_L}$$

high-frequency clock ticks during each low-frequency one. When an event of interest occurs, like the SFD event of the

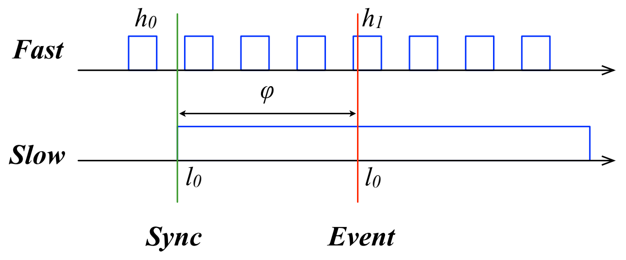

Figure 6: Illustration of the Virtual High-resolution Time event time-stamping mechanism on a micro-controller.

CC2420, the system records not only the value of the counter sourced by the low-frequency clock, but also the value of the counter sourced by the high-frequency clock which was reset at the end of the most recent clock tick of the low-frequency clock. Thus, this second timer will indicate the *phase* φ within a low frequency clock tick, allowing effective resolution to be up-sampled to the high-frequency clock (modulo one cycle of jitter). The event time is sampled as

$$t_{\text{event}} = C_L \cdot \varphi_0 + \varphi. \tag{4}$$

Generating a high-frequency clock from a low-frequency clock source is common practice. Today, most high-frequency radios synthesize a high-frequency oscillator from a typically much lower one (e.g. 2.4 GHz from 16 MHz) using a phase-locked loop (PLL) [14]. The synthesized frequency, once stable, maintains a fixed phase relation with the reference frequency. While a PLL would be an excellent choice for generating the high-frequency clock from a low-frequency on demand, PLLs are rarely found in the low-end microcontrollers that are employed in modern sensor network platforms. For that reason, we employ a second, high-frequency crystal oscillator that can be turned on or off by the microcontroller. While the high-frequency clock is not phase-locked with the low-frequency clock, the phase error is limited to $< 1/f_H$, and thus is of the same order as the quantization error.

4.1 Microcontroller-based VHT

In order to implement VHT, a microcontroller has to provide only a few standard components:

- 2 clock inputs (driven by the two oscillators), and
- 2 timers with capture and compare modes (sourced by the two clocks).

In addition, the microcontroller must either have the ability to internally enable the capture of the high frequency timer so that it is triggered by the low frequency clock signal or be augmented with external hardware that can trigger this operation. The role of the capture is to synchronize the counter values of the low- and high-frequency timers, i.e., every value captured corresponds to the time of the low frequency clock signal's rising edge. This capture value is later used to calculate the phase of an event.

Assume that two capture units, one on each timer, are connected to the SFD interrupt line of the CC2420. An additional capture unit on the fast timer captures the counter on every low frequency rising edge. This is illustrated in Figure 6 as the *Sync* event, where the value of h_0 is stored

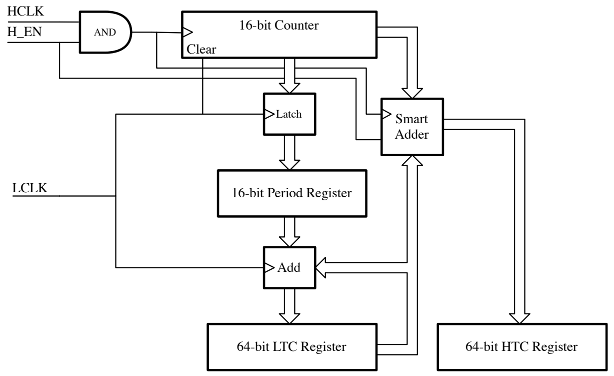

Figure 7: VHT on FPGA block diagram. The main counter has a width of 16-bit that is fed by the high frequency clock signal HCLK. The low frequency clock LCLK drives the rest of the logic and increments the different counting registers LTC and HTC based on the VHT algorithm.

in one of the capture units. Later, when the SFD line rises, the two capture units on the two timers store l_0 (the value of the low-frequency counter) and h_1 (the value of the high frequency counter), respectively. Using these three captured values, the event time can be calculated as

$$t_{\text{event}} = l_0 \cdot \varphi_0 + (h_1 - h_0) \mod \varphi_0. \tag{5}$$

The modulo operator is necessary since the microcontroller may be too slow to read the capture register value of h_0 before the next low frequency clock tick. However, since these ticks come at an interval of φ_0 high frequency ticks, the modulo operator relaxes the timing constraint.

The implementation of VHT is straightforward using a microcontroller but there are some drawbacks with certain hardware configurations. For example, our prototype platform uses the MSP430F1611 microcontroller which provides all of the necessary components. However, implementing VHT uses all of the available timer resources, impacting other components like the ADC. Since all of the timers are used, no timer is available for triggering ADC conversions that conflict with VHT usage, and thus the utility of the platform diminishes for many sensing-related operations. Another drawback is the limited width of the counters, which are only 16-bits wide. While overflows of the high-frequency counter are not a problem, overflows of the low-frequency counter must be dealt with in software to provide virtualized system counters of higher widths. In addition, while 32-bit counters may be adequate for low-frequency 32 kHz signals, such timers overflow every five minutes if they are used for VHT with a high-frequency clock of 8 MHz, suggesting that 64-bit counters will be needed to support long time intervals.

4.2 FPGA-based Dedicated VHT

Implementing VHT on an FPGA offers several advantages over a microcontroller implementation with few drawbacks. Figure 7 shows a simplified view of the major blocks contributing to a VHDL implementation of VHT. The core contains two counting registers, and one 16-bit counter. Two clock signals, the low frequency (LCLK) and the high frequency (HCLK) drive the respective parts of the implementation. An enable signal (H_EN) indicates to the timer unit if the HCLK is active or not, and switches the time resolution of the counting register HTC from high to low.

The key to the VHT implementation is a period counter register. This register stores the value of the 16-bit counter on every LCLK clock tick. Additionally, the 16-bit counter is reset to 0, and starts counting again. The period counter indicates how many HCLK ticks occur for each LCLK period, and thus indicates by how much the counting register has to be incremented at each LCLK clock tick when the HCLK clock is turned off, i.e.,

$$\text{increment} = \left\lfloor \frac{f_H}{f_L} \right\rfloor .$$

In summary, at every LCLK clock tick, an adder (Add) adds the period register to the LTC counting register. This register keeps track of time in a low-resolution mode. At the same time, a Smart Adder, depending on the H_EN signal, either adds the 16-bit counter value to the LTC counting register and stores it in the HTC counting register, or if the HCLK is turned off, just stores the LTC counting register value directly in the HTC counting register. Thus, the HTC register eventually switches between a low and high-resolution time, without losing time itself.

Two important components of a timer block are the capture and compare units, but they are just like the capture and compare registers found in microcontrollers today. The only distinction is that rather than operating on the counter that is directly incremented by the clock, they operate on the derived HTC counting register instead. The capture unit will, upon reception of a trigger signal, copy the HTC register into a timer capture register TCCR, and thus store the value for a microcontroller, which can read it later at a more convenient time. Of course, a microcontroller may need to know if the value corresponds to a high- or low-resolution timestamp. This information could easily be conveyed by one of the bits in the field, or by an additional configuration register, since the FPGA keeps track of whether it is operating in a low- or high-resolution mode.

We implemented micro-benchmarks of an initial prototype of the VHT algorithm on an Actel IGLOO AGL600 low-power FPGA. The current implementation consists of the counting mechanism illustrated in Figure 7. We also implemented a Serial Peripheral Interface (SPI) slave communication module in order to communicate with an external microcontroller. In micro-benchmarks, the FPGA core consumes 42.8 μA while the 32 kHz clock is on, and 767 μA when a 16 MHz clock drives the circuit. The quiescent current of the core itself is 36 μA during flash freeze mode, i.e., when nothing happens on the FPGA itself. We note that our current implementation demonstrates the functionality of the key VHT components but is not a fully-integrated, stand-alone VHT implementation. For example, our current design does not include working capture and compare units, which we leave to future work.

Unfortunately, even the FPGA implementation of the VHT algorithm has some drawbacks. In general, the power draw will be higher than if it were directly integrated into a microcontroller. Additionally, the fact that the timers are off-chip necessitates a communication interface between the microcontroller and the FPGA. While SPI can operate at frequencies up to 10 MHz, it takes time to setup and retrieve the data, which can be cumbersome and sometimes even prohibitively slow depending on the application requirements. In an ideal world, our partial FPGA implementation would be put onto silicon and directly integrated into a microcontroller. This would achieve the best of both worlds: large timers and high speed for the measurements of the phase, and low-power due to the direct integration in silicon.

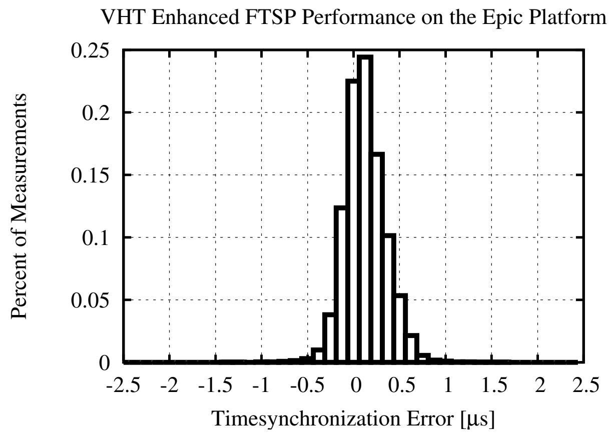

Figure 8: Distribution of VHT synchronization error. The high-frequency 8 MHz virtual clock provides a maximum time-resolution of 0.125 μs. Using this clock, we achieve an average accuracy of 0.125 μs (one tick) with a standard deviation of 0.645 μs. These results break new ground.

5. EVALUATION

In this section, we evaluate the performance of the microcontroller implementation of VHT since, with only minor hardware modifications, it represents the most likely adoption scenario on other platforms. We implement these modifications on the Epic Core [15], a modularized mote that is supported by TinyOS and integrates both the CC2420 radio and the MSP430F1611 microcontroller. However, the Epic Core does not provide an external high-speed crystal, and the MSP430's internal RC oscillator does not provide enough stability for the microcontroller VHT implementation. Therefore, we add an external hardware clock that we can enable and disable using a GPIO pin of the microcontroller. The only other hardware modification necessary is a connection from the SFD interrupt to the second timer unit capture input. Fortunately, both of these are simple modifications, since the Epic Core exposes both the SFD line and connections for a secondary crystal on its perimeter pads. Therefore, we can simply feed the SFD signal back into an additional capture input and add a high-frequency crystal. We produced five Epic Core nodes modified in this manner for evaluation, limiting our sample size to five.

A number of software modifications are needed to evaluate VHT as well. We start with the standard TinyOS 2.x release, including the FTSP protocol. We modify TinyOS to configure both hardware timer units of the MSP430 microcontroller. In addition, we make some small modifications to the radio stack so that an incoming or outgoing

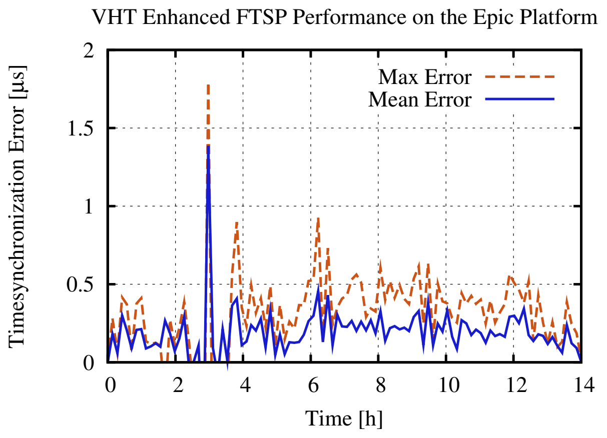

Figure 9: VHT synchronization error over 14 hours with a resynchronization interval of 10 seconds.

radio message is timestamped with not just the value of the 32 kHz clock, but also with the phase φ from the secondary (high-frequency) timer unit as well. These modifications are backward-compatible and they make it possible to synchronize a normal node that cannot support VHT with a node that does support VHT. However, this backward compatibility only works if the VHT node is the reference, but not vise versa, since VHT relies on the reference for estimating other nodes' phase offset.

To evaluate the accuracy of VHT, we begin with the five VHT-capable nodes synchronizing at an interval of 10 seconds. A sixth node sends out a beacon at two second interval, which the five VHT nodes timestamp. These timestamps are collected through a wired back-channel in order to minimize packet collisions and message losses. Figure 8 presents the histogram of the measurements, while Figure 9 illustrates the mean and maximum error over the 14 hour long experiment. The histogram shows a Gaussian distribution with a mean of 0.125 μs and a standard deviation of 0.625 μs. To the best of our knowledge, this level of synchronization accuracy has not been achieved before in a low-power wireless sensor network.

We next explore the veracity of our earlier claim that VHT offers high-accuracy time synchronization but at potentially reduced power draw, since VHT duty cycles the high-frequency crystal during sleep intervals. We claimed this allows VHT to achieve a baseline power draw that converges with that of 32 kHz sleep timer at low duty cycles.

To obtain the power data, we hand-modify a TMote Sky rather than use the modified Epic Core. This rationale for this decision is that we wish to characterize the power draw one would expect in a real setting on a widely-adopted platform rather than the power draw of a prototype platform with a range of (other) peripherals that are not power-optimized.[6] To exclude the influence of all these additional power sinks, we hand-modify a TMote Sky by adding an external 8 MHz crystal and a patch wire from the CC2420

[6]As is the case with our Epic Core-based test nodes that include a host of other sensors, signal conditioning circuitry, and peripherals not present on the TMote or most other sensor nodes.

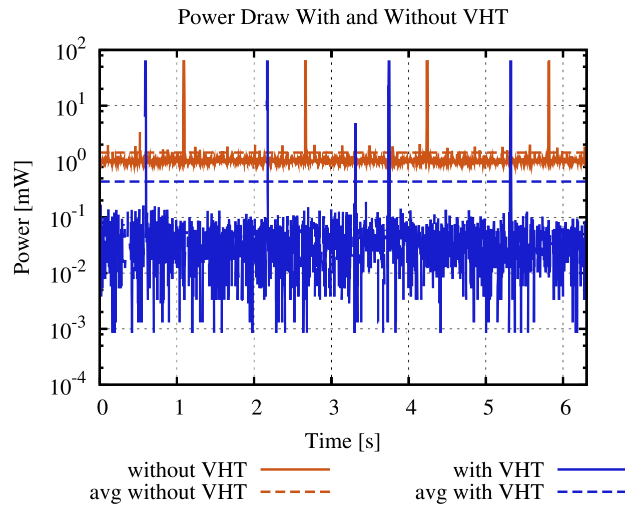

Figure 10: Comparison of the power draw of regular and VHT-enhanced nodes. Both systems run at 3V. The power is measured on an enhanced version of a TMote Sky using TinyOS 2.x and the default Low-Power Listening MAC with a polling interval of 1.6 s, corresponding to a duty-cycle of 0.77%. The system using VHT draws and average power of 437 μW and the system without VHT 1449 μW, or three times greater. Note that at 0.77% duty-cycle, radio communication draws on average 427 μW, thus dominating the power budget over the VHT clock system by a factor exceeding 40.

P_0	P_{lclk}	P_{hclk}	P_{Radio}
4.20 μA	1.06 μA	340.1 μA	18,866 μA

Table 1: Power draw of various VHT subsystems.

SFD line to an available Timer A capture pin.

Figure 10 shows the power draw of the modified TMote Sky when running FTSP using the 8 MHz crystal as the sole clock source, and also when running with VHT as a clock source. Note the large dynamic range in power draw for VHT (five orders-of-magnitude) poses a significant challenge to the Agilent digital multi-meter (DMM) used for these measurements. The mote running VHT approaches the noise floor of the measurement instrument. [7] In these experiments, the node is running the default CC2420 Low-Power Listening MAC protocol in TinyOS and the polling interval is set to 1600 ms. Therefore, the radio is turned on to take a channel sample every 1.6 seconds; these channel samples are the large spikes visible in the power draw.

Since VHT effectively duty-cycles the high-frequency clock along with the radio, its power draw is approximately 3$\times$ lower than without VHT, as expected. Extrapolating from this observation, we create a power draw model for a VHT-equipped node. Given the duty cycle dc, we know that the

[7]We verify the power draw during sleep using different DMM settings. However, in that mode the active radio power draw is beyond the measurement range and thus invalidates average power draw calculations.

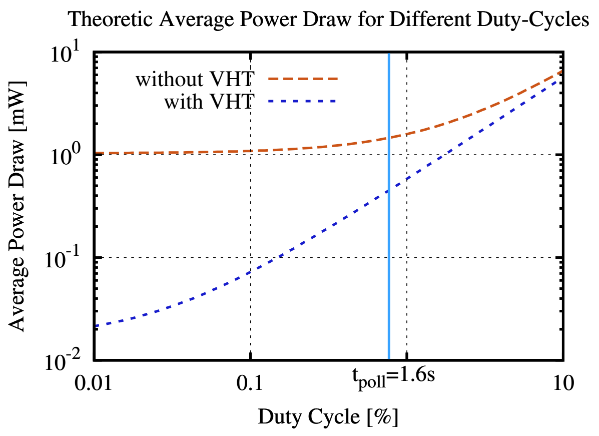

Figure 11: Theoretic average power draw of a duty-cycling node with and without VHT. The lower average power for the VHT equipped node comes from the duty-cycling of the high frequency clock. Note that starting at a duty-cycle of below 0.1%, the power improvement is $> 10\times$.

average power draw of a node running LPL is given by

$$P_{\text{avg}} = P_0 + P_{hclk} + dc \cdot (P_{\text{radio}}) \tag{6}$$

for the regular node, and

$$P_{\text{avg,VHT}} = P_0 + P_{lclk} + dc \cdot (P_{\text{hclk}} + P_{\text{radio}}) \tag{7}$$

for the VHT enhanced node, where P_0 is leakage, P_{lclk} the slow clock, P_{hclk} the fast clock, and P_{radio} the radio power draw. We can compute the particular power draws from the data in Figure 10 given the LPL setting of 1.6 s corresponds to a duty-cycle of 0.77%. Table 1 summarizes these results, and Figure 11 illustrates the two average power draws, showing where the model and empirical data intersect. VHT power draw scales well with duty cycle but provides the benefits of high-precision time-keeping. The figure shows the power-proportionality of VHT: as the duty cycle decreases, the power draw of the VHT-equipped node approaches that of a node clocked from just a 32 kHz crystal.

6. RELATED WORK

Time synchronization protocols and sensor network applications have a long and tangled history. In this section, we highlight several important contributions to the time synchronization literature, but our focus is primarily on algorithms and systems that have been tested on similar hardware as we employ on our prototype hardware.

Reference Broadcast Synchronization (RBS) [7] was one of the first time synchronization protocols proposed for sensor networks. RBS' main contribution is the removal of the transmitter's nondeterminism from the communications critical path. RBS addresses the problem that it was difficult to accurately timestamp outbound messages due to queueing and other delays, but that two receivers of a single message implicitly share a point of reference against which to compare their clocks. While this approach is beneficial in the face of high transmission nondeterminism, it becomes less relevant when low-level software- or hardware-based times-tamping mechanisms are available. As we show in Section 2.2.2, the CC2420 and RF230 radios offer just such support, and are able to accurately and precisely time-stamp a radio message. The CC2420 is so accurate, in fact, that error sources not considered relevant by RBS become important. RBS, for example, ignores the propagation delay of messages. While this is not a problem if time-stamping precision is worse than about 1 μs, it starts to be a significant source of error at appreciably finer precisions. For example, if two receivers are separated by a distance greater than 30 meters, then an error of 100 ns may be introduced in the common reference. A system using RBS could improve its power/performance by using VHT, but it would still encounter inaccuracies at sub-microsecond levels.

The Timing-sync Protocol for Sensor Networks (TPSN) [8] uses a simple two-way synchronization exchange in order to remove sender and receiver uncertainties, and compensate for the propagation delay. The only sources of error remaining are variation in the time-stamping mechanism and clock drift during the synchronization exchange. Through some analysis, the authors of [8] show that, in theory, TPSN should have a $2\times$ better performance than RBS. However, the authors do not implement a clock drift estimation technique, so their actual synchronization error is about 20 μs, which is high compared to newer synchronization protocols. TPSN would benefit greatly from using VHT, and by employing clock drift estimation, TPSN's prospects for the next generation of high-accuracy time synchronization protocol are excellent. IEEE 1588, the Precision Time Synchronization Protocol that runs over Ethernet, achieves accuracies of better than 100 ns using a technique very similar to TPSN.

The Flooding Time Synchronization Protocol (FTSP) [5] is the *de facto* time synchronization protocol in sensor networks. A key contribution of FTSP was to exploit the TI CC1000 radio's byte interrupt for a high-precision message time-stamping. In addition, its simple and efficient time dissemination algorithm and root election protocol make it a robust approach to synchronizing an entire network of nodes to the time of one root node. While subsequent research improved FTSP's startup time (time to initial synchronization) [16] and multi-hop performance [17] with only minimal changes to the FTSP algorithm [10], no other work has fundamentally improved on FTSPs time-stamping, power, or per-link accuracy. While we use FTSP in our evaluation of VHT, we note that FTSP suffers from the same drawbacks as RBS. First, FTSP does not account for propagation delays, which become a non-trivial factor once nodes are separated by distances exceeding 30 meters.

A challenge in deep multi-hop time synchronization networks are nodes that are radio neighbors, but members of two different synchronization trees. The error propagation in the two synchronization trees is likely to be different, and so the two radio neighbors can have a potentially large difference in their views of global virtual time. This situation can become problematic if the synchronization protocol is used for synchronized communications, as might be the case for a scheduled MAC protocol. The Gradient Time Synchronization Protocol (GTSP) [6] offers one solution to this problem. GTSP synchronizes a logical local clock to the radio broadcasts overheard from all the neighbors, resulting in nearby neighbors being in much closer synchronization. While GTSP would benefit from the power efficiency of VHT, it is unclear to us how GTSP deals with propagation errors since the GTSP paper does not explore the issue in its evaluation [6]

VHT with TCTS is unusual, but not unique, in its focus on the power cost of accurate time. Rowe et. al present the Syntonistor, a new device for time synchronization that phase and frequency locks onto ambient 60 Hz AC noise found in nearly all buildings [18]. The device includes a tuned LC tank circuit that resonates at 60 Hz and small microcontroller that uses a PLL for robustness. Using this approach, Rowe et al. showed that over an 11 day period, the accuracy of their device was on average better than 1 ms, while drawing only 58 μW. While not of comparable accuracy, the Syntonistor's low average power draw makes it an attractive choice. Whether the Syntonistor's accuracy can be extended using a technique like VHT, to provide high-precison during times of need, while relying on the Syntonistor for long-term stability, remains an open question.

The most similar work to the VHT algorithm is the Harmonia time-synchronization system [19]. Harmonia is a system in which a TCXO-driven RTC that provides a stable 1 Hz signal to a node is synchronized to an external reference. The Harmonia authors argue that while the 1 Hz signal is quite stable, an ambiguity of at least 1 clock tick is always present. Therefore, they use the high-frequency microcontroller clock that exists on every node to synchronize the external RTC signal. The main distinction between Harmonia and a regular time synchronization is that Harmonia does not need to concern itself with frequency error estimation because Harmonia relies on the presence of a stable external clock at every node. Thus, Harmonia only performs offset calibration and therefore may not be widely applicable to other system architectures. In contrast, VHT offers a drop-in replacement for high-frequency clocks in any system architecture, regardless of whether an external RTC or just a simple 32 kHz crystal is present.

7. CONCLUSION

Time synchronization is a fundamental service that is used by all layers of system and application software, from scheduling link layer communications to time-stamping application layer events. However, sensor network designers have historically faced a dilemma with time-keeping and synchronization: they could choose either low-power or high-resolution, but not both, for the simple reason that higher temporal resolutions translate to higher clock frequencies, which in turn translate to higher power draws.

In this paper, we present Virtual High-resolution Time, a new approach to time-keeping that offers both low-power and high-resolution by using two clocks – one slow (and low-power) and one fast (and high-resolution) – and carefully coordinating their usage to present the illusion of a single high-precision, power-proportional clock. VHT power draw scales with clock access (i.e. reading and writing) rather than operation (i.e. on or off), allowing applications to spend energy in proportion to their actual needs.

We present a VHT implementation using an off-the-shelf microcontroller and sketch a design using an FPGA that shows how it can be integrated into future silicon. Our results show that VHT offers better than 1 μs time-keeping precision and an order-of-magnitude improvement in power draw compared to conventional techniques at 0.1% or lower radio duty cycles. These results enable previously infeasible application regimes for timing-critical sensor networks.

Acknowledgments

This material is supported in part by the the U.S. ARL and the U.K. MOD under Agreement Number W911NF-06-3-0001, NSF Award Number CCF-0820061, and by UCLA's NSF Science and Technology Center for Embedded Networked Sensing. Thomas Schmid is currently funded by a fellowship from the Swiss National Science Foundation. This work was supported in part by both NSF and Microsoft Research Graduate Fellowships. 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.

8. REFERENCES

[1] B. Van Veen and K. Buckley, "Beamforming: A versatile approach to spatial filtering," *IEEE ASSP Magazine*, vol. 5, no. 2, pp. 4–24, 1988.

[2] E. Candes and M. Wakin, "People hearing without listening: An introduction to compressive sampling," *IEEE Sig. Proc. Magazine*, 2007.

[3] H. Wang, L. Yip, D. Maniezzo, J. Chen, R. Hudson, J. Elson, and K. Yao, "A wireless time-synchronized COTS sensor platform: Applications to beamforming," 2002.

[4] P. Dutta, D. Culler, and S. Shenker, "Procrastination might lead to a longer and more useful life," *HotNets-VI*, 2007.

[5] M. Maróti, B. Kusy, G. Simon, and Á. Lédeczi, "The flooding time synchronization protocol," *SenSys*, pp. 39–49, 2004.

[6] P. Sommer and R. Wattenhofer, "Gradient clock synchronization in wireless sensor networks," in *IPSN*, 2009.

[7] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 147–163, 2002.

[8] S. Ganeriwal, R. Kumar, and M. Srivastava, "Timing-sync protocol for sensor networks," *SenSys*, 2003.

[9] Texas Instruments, "CC1000 product information." http://focus.ti.com/lit/ds/symlink/cc1000.pdf.

[10] C. Lenz, P. Sommer, and R. Wattenhofer, "Optimal clock synchronization in networks," in *SenSys*, ACM, 2009.

[11] J. Polastre, R. Szewczyk, and D. Culler, "Telos: enabling ultra-low power wireless research," in *Proceedings of the 4th international symposium on Information processing in sensor networks*, 2005.

[12] T. Schmid, Z. M. Charbiwala, R. S. Shea, and M. B. Srivastava, "Temperature compensated time synchronization," *IEEE Embedded Systems Letters*, 2009.

[13] J. W. Hui and D. E. Culler, "Ip is dead, long live ip for wireless sensor networks," in *SenSys*, (New York, NY, USA), pp. 15–28, ACM, 2008.

[14] D. Wolaver, *Phase-locked loop circuit design*. Prentice Hall, 1991.

[15] P. Dutta, J. Taneja, J. Jeong, X. Jiang, and D. Culler, "A building block approach to sensornet systems," in *SenSys*, 2008.

[16] B. Kusy, P. Dutta, P. Levis, M. Maroti, A. Ledeczi, and D. Culler, "Elapsed time on arrival: a simple and versatile primitive for canonical time synchronisation services," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 1, no. 4, pp. 239–251, 2006.

[17] J. Sallai, B. Kusy, A. Ledeczi, and P. Dutta, "On the scalability of routing integrated time synchronization," *EWSN*, 2006.

[18] A. Rowe, V. Gupta, and R. Rajkumar, "Low-power clock synchronization using electromagnetic energy radiating from ac power lines," in *SenSys*, ACM, 2009.

[19] J. Koo, R. K. Panta, S. Bagchi, and L. Montestruque, "A tale of two synchronizing clocks," in *SenSys*, 2009.