Hardware Assisted Clock Synchronization for Real-Time Sensor Networks

Maxim Buevich, Niranjini Rajagopal, Anthony Rowe
Electrical and Computer Engineering Department
Carnegie Mellon University
Pittsburgh, USA.
{mbuevich, niranjir, agr}@ece.cmu.edu

***Abstract*—Time synchronization in wireless sensor networks is important for event ordering and efficient communication scheduling. In this paper, we introduce an external hardware-based clock tuning circuit that can be used to improve synchronization and significantly reduce clock drift over long periods of time without waking up the host MCU. This is accomplished through two main hardware sub-systems. First, we improve upon the circuit presented in [1] that synchronizes clocks using the ambient magnetic fields emitted from power lines. The new circuit uses an electric field front-end as opposed to the original magnetic-field sensor, which makes the design more compact, lower-power, lower-cost, exhibit less jitter and improves robustness to noise generated by nearby appliances. Second, we present a low-cost hardware tuning circuit that can be used to continuously trim a micro-controller's low-power clock at runtime. Most time synchronization approaches require a CPU to periodically adjust internal counters to accommodate for clock drift. Periodic discrete updates can introduce interpolation errors as compared to continuous update approaches and they require the CPU to expend energy during these wake up periods. Our hardware-based external clock tuning circuit allows the main CPU to remain in a deep-sleep mode for extended periods while an external circuit compensates for clock drift. We show that our new synchronization circuit consumes 60% less power than the original design and is able to correct clock drift rates to within 0.01 ppm without power hungry and expensive precision clocks.**

Keywords*—*Time Synchronization; Clock-rate Adjustment; Wireless Sensor Networks;

I. Introduction

Time synchronization is a vital service in distributed systems that enables event ordering and coordinated communication. These are both critical aspects for improving system Quality-Of-Service (QoS) required by real-time applications. In wireless sensor networks (WSNs), time synchronization is particularly challenging because of the uncertainty and jitter associated with the arrival of external timing information. Packet arrival times can be averaged to reduce these inaccuracies, but such methods increase energy consumption. A significant body of previous work exists on minimizing this communication overhead while reducing uncertainty in timing. Most of these approaches rely on compensating for clock drift rates using software adjustments. In this paper, we present a clock synchronization expansion module for the *FireFly3* WSN platform that maintains clock synchronization through a novel hardware clock rate adjustment module. Shifting the rate-adjustment burden closer to the hardware layer results in two advantages. First, it enables the system to adjust for clock drift in a more continuous fashion that is able to reduce error more effectively than software solutions. Second, the approach offloads synchronization burden from the host micro-controller allowing it to remain in a deep-sleep mode of operation for extended periods without having to periodically wake up to adjust for drift.

Crystal oscillators used in micro-controllers rely on the mechanical resonance of a vibrating quartz element to create a periodic electrical signal. In sensor networks, there is typically a low-frequency clock that operates continuously even when the processor is in a deep-sleep power state. Once the processor wakes from sleep, a high-speed (and more energy consuming) clock is activated to drive the instruction counter on the MCU. Clocks on different nodes will suffer from frequency variation due to factors such as temperature differentials, manufacturing limitations, aging effects and mechanical acceleration. These factors cause each node's notion of time to drift with respect to other nodes. In order to compensate for this drift, the system requires a global timing reference.

The most common WSN approach for providing a global timing reference is through message passing. Protocols like [2], [3], [4] exchange packets in a manner that allows them to reduce timing uncertainties. In [1] we presented a hardware device called a *Syntonistor* that is able to achieve a similar goal by tracking the 60 Hz magnetic field (common across a wide area) being emitted from power lines within a building.

The power line signal can be used as a global clock source for battery-operated sensor nodes to eliminate drift between nodes over time even when they are not passing messages. With this scheme, each receiver is frequency-locked with each other, but there is typically a phase-offset between them. Since these phase offsets tend to be constant, a higher-level compensation protocol can be used to globally synchronize a sensor network (described in more detail in [1]). The first contribution of this paper is an improved Syntonistor sensing front-end that is based on electric field sensing, as opposed to magnetic resonance. In Section IV we show that the new circuit reduces power (less than half on average), cost, complexity and provides an input with lower jitter (more than 75% better) than the previous design. Once a system has an available global time reference, the host micro-controller is required to periodically resynchronize with it in order to account for clock drift. At any given moment, the accuracy depends on the clock's drift rate and the last synchronization point. This implies that in order to achieve tighter synchronization the system must perform wake up operations more frequently. These wake up operations have

an associated energy penalty. We propose and evaluate a rate adaptive software scheme to help reduce this overhead.

The second contribution of this paper is a novel hardware clock-rate adjustment module that allows an external ultra-low-power device to continuously tune the main processor's clock at runtime. Such an approach is often used by temperature controlled oscillators, which employ a control loop to tune a voltage-controlled oscillator (VCO), thereby maintaining more accurate time. In our case, the input to the control loop is a global clock reference being supplied by the syntonistor circuit. This can entirely remove the need for periodic resynchronization wakeups of host processor. As compare to our rate adaptive software scheme, hardware clock tuning has the added benefit of simplifying the software running on the host. In the case of the FireFly3, we use the Syntonistor circuit to control the host CPU's clock, however the same principle can be applied using other external synchronization sources (for example WWVB receivers).

Our rate adjustment circuit uses a varactor (variable capacitance diode) in place of one of the load capacitors typically found on low-power clock crystals. This allows a programmable digital-to-analog converter (DAC) to push and pull the oscillator's frequency. By carefully adjusting the load capacitance, we can safely tap one side of the crystal's output and feed it into the input of an external microcontroller that monitors the crystal's rate. In the FireFly3 hardware design, the MCU, which tracks the 60Hz power line signal, also senses and tunes the main clock driving the FireFly3 node. This allows the Syntonistor to autonomously adjust the host MCU clock to compensate for drift. This has two main benefits: (1) the main host can now sleep deeply uninterrupted for extended periods of time and (2) if asynchronous events wake the node, they can be immediately and accurately timestamped without the need for resynchronization.

The remainder of the paper is organized as follows. Section II will review work related to WSN time synchronization and hardware clock rate adjustment. Section III will discuss clock synchronization using the Syntonistor and hardware rate adjustment. Section IV will discuss the details of our system design including our improved Syntonistor and our hardware clock rate adjustment circuit. Section V will provide experimental performance results and Section VI will discuss limitations of the system. Finally, Section VII will conclude and provide future work.

II. Related Work

A significant amount of work from the distributed systems community can be applied to wireless sensor networks. [5] presents the seminal work on *logical clocks* for total event ordering in a system. The approach captures a numerical method for ordering events based on time-counter exchanges among devices. More recent work has focused directly on establishing a common notion of wall-clock time [6], [7], [8], [9]. The most commonly adopted of these approaches is the Network Time Protocol (NTP) that uses round-trip message delay averaging to set times. When changes to time are required, NTP uses a clock-rate adjustment technique ensuring that time remains continuous. For example, if a computer's notion of time falls behind actual time, it adjusts its mapping of ticks to seconds such that its virtual clock runs faster. We naturally use a similar approach since our hardware clock-rate adjustment circuit will gradually adjust the clock without causing discrete "jumps" in the main CPU's notion of time.

Next, we look at message passing approaches for synchronization in wireless sensor networks. Many of these approaches achieve extremely accurate synchronization, but few evaluate the required energy associated with maintaining synchronization. The reference broadcast synchronization [3] (RBS) scheme uses timestamps exchanged between multiple receivers to eliminate transmission delays. This approach specifically targets the sources of timing jitter associated with wireless devices and averages over multiple transmissions to achieve tight pairwise clock synchronization. The flooding time synchronization protocol [2] and the time-sync protocol for sensor networks [10] (TPSN) use low-level hardware timestamping to eliminate these similar sources of timing jitter. Messages are flooded across the network forming a spanning tree that periodically compensates for drift. Local clock rates are adjusted to help reduce drift, which could also be achieved using our module. Software approaches have also proposed mixing high-speed and low-speed clocks together to improve timestamp accuracy in a low-power manner [11]. In Section III we discuss the trade-offs associated with hardware and software approaches to clock rate adjustment. In [12], the authors propose an interesting approach inspired by fireflies and other biological systems that allows groups of nodes to achieve synchronicity through rate adjustment of message passing with their neighbors. In [4], the authors propose a scheme called Glossy, which utilizes constructive interference at the symbol level to boost the probability of successful time synchronization messages. All of these approaches could be used as part of the initial synchronization mechanism required by the Syntonistor [1].

Multiple synchronization approaches leverage external hardware to receive global time broadcasts. The WWVB atomic clock broadcast from NIST uses a 50kW radio tower located in Boulder, Colorado to transmit a 60Khz time beacon. This system encodes wall-clock time using a pulse width modulated coding scheme with rising edges at the beginning of each second. This is ideal for outdoor applications within the tower's broadcast range, but the radio transmission does not penetrate far into buildings. We could easily see this approach being used as an alternative mechanism to our 60Hz receiver for outdoor applications. The Global Positioning System (GPS) uses precise clock synchronization derived from satellite transmissions for localization. GPS time receivers have commonly been used as sources for NTP servers and often use temperature-controlled oscillators to improve timing stability. These have even been implemented in software for wireless sensor networks [13]. Temperature controlled oscillators internally tune crystals in a similar fashion to what we use in our approach. Our approach however works well with common clock configurations with the addition of just a few simple components. The Radio Data System (RDS) uses the sidebands on standard FM radio transmissions to encode data, including the time. These receivers typically consume too much energy for use on a node-by-node basis in a sensor network, and in the case of GPS and WWVB require direct line of sight with the sky. The RT-Link [14] protocol uses a carrier current AM radio transmitter to send global time beacons to sensor nodes. The

system uses a building's wiring infrastructure as an antenna to broadcast an AM radio timing pulse to an external receiver on all nodes. This solution works well for industrial applications, but it requires a centralized radio transmitter that is expensive and often difficult to install. The work in this paper focuses on how best to couple timing inputs to a hardware system in order to allow for extended and uninterrupted deep-sleep periods.

III. Clock Synchronization

Two clocks are considered synchronized when their deviation from each other is bounded by some known value. Even if two real clocks are set to identical values simultaneously, they will slowly deviate due to "clock drift" resulting from minor variations in clock frequency. In order for a system to remain synchronized, a global clock signal must be present from which each device estimates its internal drift. The work in this paper builds upon the device described in [1] (called a *Syntonistor*) that recovers a common global clock from the electromagnetic field generated by power lines. Since each node in the system can lock onto the same global clock source, the nodes can remain synchronized for long periods of time without exchanging messages. This work also shows that the synchronization accuracy in such a system is determined by the amount of jitter in the global clock source and not neccessarily the frequency.

In the original system, the Syntonistor device generated a Pulse-Per-Second (PPS) output that was read by the host sensor node's processor to remain synchronized with the network. To remain synchronized, the host was required to wake up and record each PPS edge so as to avoid drifting away from the global time. Even though the Syntonistor itself is quite low-power (drawing $58uW$ of power), there is a significant amount of overhead required to periodically wake up each second and synchronize two nodes. One possible solution to this problem would be to connect the PPS line from the Syntonistor to an external counter, enabling the host processor to keep track of seconds while in deep-sleep mode. Alternatively, if a counter is not available, the processor could be scheduled to wake up before the worst-case clock drift would cause it to miss a PPS edge. While a significant improvement upon the approach adopted by the original Syntonistor, this type of approach still suffers from an energy overhead and makes it difficult to timestamp asynchronous events. For example, if a node is in deep sleep for an extended period of time and an interrupt triggers an event; the node internally could have drifted by up to the PPS input period. The node would need to wait until the next period edge to fully resynchronize and then back-compute the event's exact timestamp. For low-latency applications this delay could be problematic.

In this paper, we introduce the FireFly3 WSN platform with a hardware clock rate adjustment module that is independently controlled by an improved Syntonistor to adjusts the local sensor's notion of time. This means that the sensor node can remain synchronized for long periods without ever communicating directly with the Syntonistor (or even waking up from deep sleep). Any firmware running on the sensor node would transparently benefit from the synchronization infrastructure. Since the new Syntonistor can continuously adjust the clock rate, it can also improve synchronization accuracy by reducing clock rate tracking error.

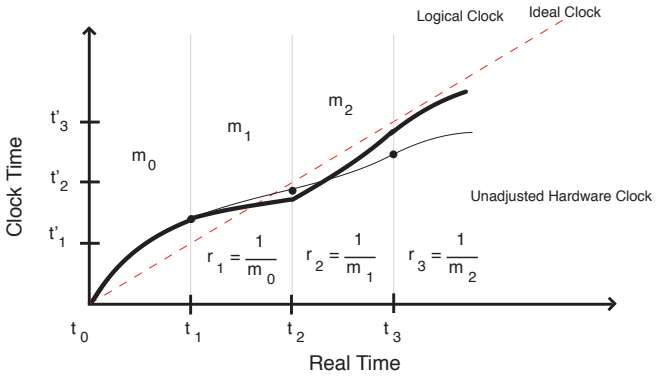

Fig. 1. Clock rate adjustment at a low sampling rate

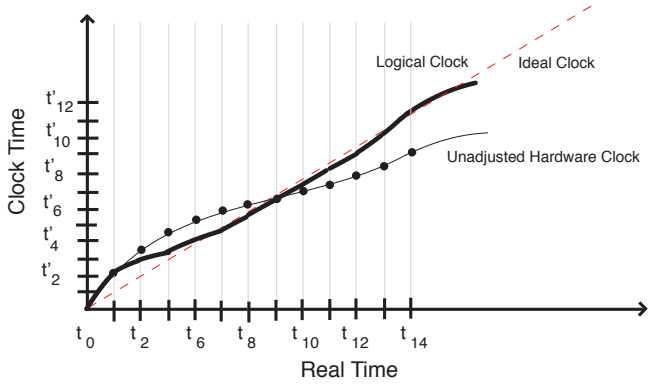

Fig. 2. Clock rate adjustment at a high sampling rate

Figure 1 shows an example of a hardware clock drifting with respect to real-world time. In order to highlight the benefits of continuous rate adaptation, we model the hardware clock as a non-linear function. In reality, since clock performance varies due to environmental parameters such as temperature and vibration, clocks exhibit non-linear behaviors over various time scales. For example, a sensor node that cycles between hot and cold over a 24-hour period as the sun rises and sets would appear to have a non-linear clock rate if sampled every hour. Alternatively, one could imagine the clock used for a tire pressure monitoring system that is rotating inside a car's wheel and being subjected to constantly changing acceleration. Since oscillators are mechanical devices, this would cause rapid fluctuations in clock frequency on the order of tens or hundreds of ppm. The x-axis of Figure 1 shows real world time. The y-axis shows the clock's notion of time. An ideal clock is drawn with the dashed line where the slope (depending on axis scaling) will be 1. The thin curve above the ideal clock (exaggerated for illustration purposes) represents a non-linear hardware clock. Adjustments are made at sampling instances denoted by the time markers on the x-axis. For example, a clock correction update occurs at time t_0, t_1, t_2 and t_3. We employ a rate adaptation approach similar to the one described in [15], where at each time-instant an error slope is computed and is subsequently used for correction during the next time-instant. For example, between t_n and t_{n+1} we compute slope m_n and apply a rate $r_n = \frac{1}{m_{n-1}}$. The difference between m_n and the ideal clock is the rate error over the last period

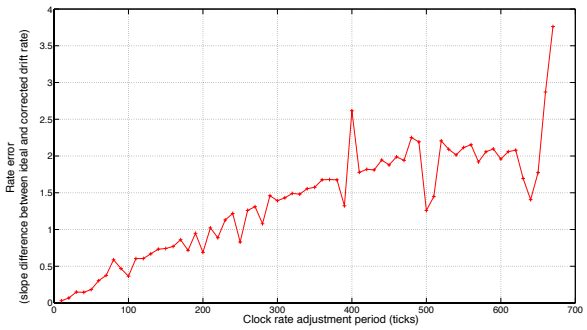

Fig. 3. Tracking error vs sample-rate

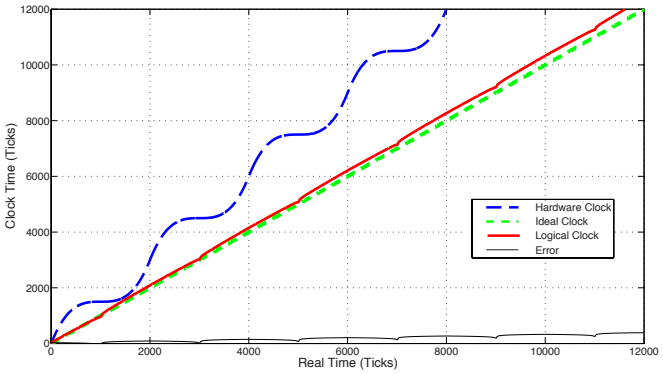

Fig. 4. Clock rate adjustment period of 10 ticks

of time. We then apply a rate adjustment $r_{n+1} = \frac{1}{m_n}$ to the next time step t_{n+1} to t_{n+2} using our tunable hardware clock. Logical time adjustments can be achieved in software or hardware. Since the real hardware clock can potentially change rate during the next sample periods, we see that the adjusted clock does not perfectly reach the ideal clock line. In Figure 2, we see an equivalent scenario, but with a shorter sampling interval, and better tracking performance. In general, intuition would suggest that faster sampling rates lead to more accurate tracking and result in less average error. It is interesting to note that this is not always the case.

To quantify the impact of sampling rate on tracking error,

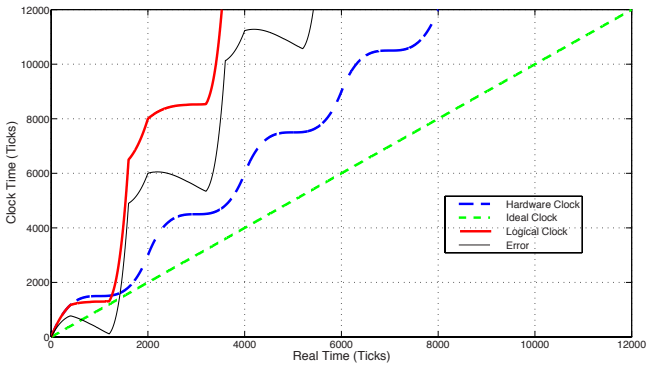

Fig. 5. Clock rate adjustment period of 400 ticks

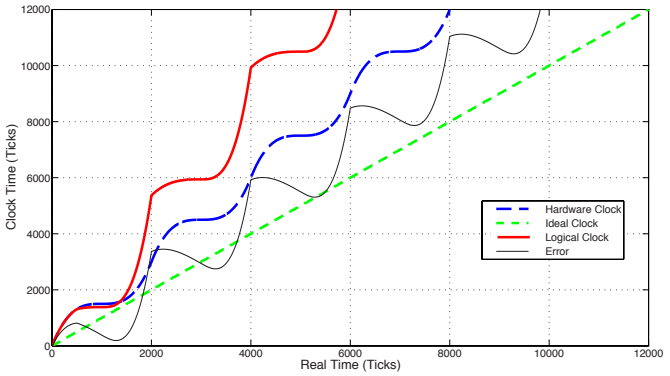

Fig. 6. Clock rate adjustment period of 500 ticks

we simulate our rate adjustment algorithm on a hardware clock that uses a monotonically increasing function as the underlying drift model. The key feature that impacts a clock's ability to track ideal time is the rate at which the clock deviates as compared to the clock adjustment sampling-period. In Figure 3 we show the impact of the sampling period on clock drift rate over time. The x-axis shows the clock sampling period in ticks. A lower tick sampling value indicates that the clock is being adjusted more frequently (faster). The y-axis shows the rate error between adjusted time and real time. Of note here is the fact that, as the sampling period gets smaller, the tracking error decreases. This indicates that continuous tracking will tend to perform better than discrete tracking. However, we do see certain operating points where faster sampling rates lead to worse tracking performance. This is an artifact of sampling when the slope is nearly zero and applying a high rate correction for the next period. Figure 4 shows the best tracking performance when the sampling rate is as high as every 10 ticks. Figure 6 shows an example where a slower sampling rate (per 500 ticks) performs better than that found in Figure 5 which is operating at a faster rate (per 400 ticks). We cannot guarantee that sampling faster always improves accuracy, but in general it tends to perform better.

IV. System Design

We now describe the hardware architecture and details of each component in our system. At the core of the FireFly3 wireless sensor networking platform is an Atmel ATmega128rfa1 8-bit System-On-Chip (SoC) microcontroller with integrated 802.15.4 radio. The CPU has 16KB of RAM and 128KB of flash memory. A 16MHz clock drives both the radio transceiver and the main CPU when it is in the active and idle mode of operation. A 32KHz low-power clock is used to maintain global time and runs continuously even when the system is in deep sleep mode. In addition to the main processing and communication unit, there are two additional novel components used to improve timing: (1) an updated power-line sensing synchronization circuit and (2) a hardware clock-rate adjustment circuit. Figure 7 shows a photograph of the FireFly3 sensor node next to the updated electric field sensing board. The remainder of this section will discuss the design choices and evaluation of these two modules.

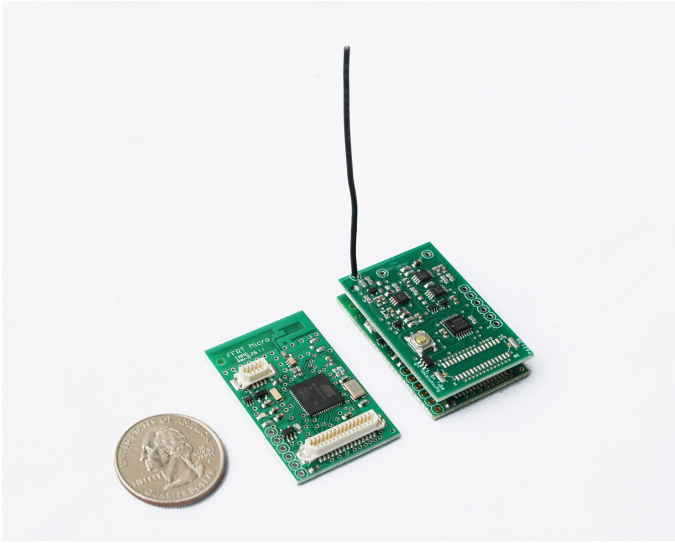

Fig. 7. FireFly3 sensor node (left) and FireFly3 with Electric Field sensor stacked on top (right)

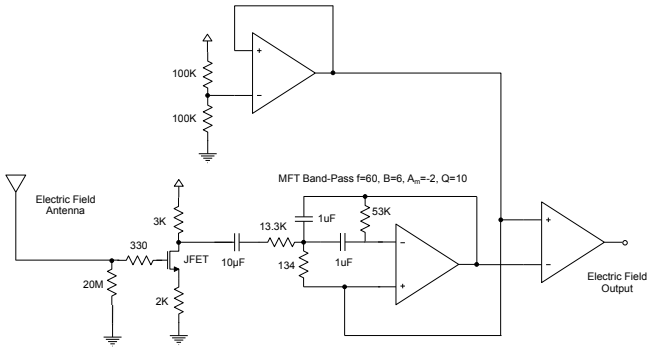

Fig. 8. Electric field detector circuit

A. Electric Field Sensing

In [1] we proposed a synchronization module that locks onto the 60Hz magnetic fields generated from power lines in order to capture a common global time reference. The original design detected magnetic fields using an inductive pickup. This design was originally chosen because of its high sensitivity and hence long detection range. Magnetic fields are generated by the flow of current, which means the circuit will only detect the signal when appliances are active in a building. On the other hand, electric fields are generated from a voltage potential. This means that the module can lock onto a signal even if no current is flowing. For example, the wires from electrical mains that go to light switches still generate a detectable electric field when the light is switched off. It also makes the system less sensitive to nearby appliances turning on or off since this will no longer alter the intensity of the signal. The circuit is also less complex, lower cost (no longer requiring a large coil) and consumes less power. The main drawback of using an electric field sensing approach is that the sensitivity is lower as compared to the magnetic field front-end. In practice, the sensitivity can easily be increased by adding a longer antenna.

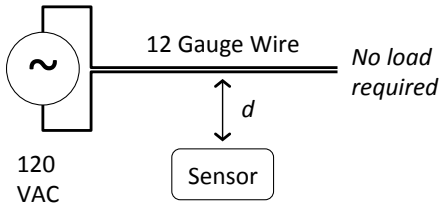

Fig. 9. Electric field experimental setup

Figure 8 shows the updated electric field sensing circuit. The main sensor front-end uses a JFET and a small wire acting as a Hertzian antenna to detect potential differences across an electric field. The JFET opens or closes based on the change in force exerted by the electric field. The large-valued resistor between the gate and ground acts as a runoff to remove excess charge buildup from constant nearby fields. The output of the FET is then passed through a second-order multiple feedback topology band-pass filter tuned to 60Hz. As compared to the design in [1], this additional filter significantly reduces spurious noise that is not generated by the appliance. The output of the electric field sensor is passed to an MSP430G2452 low-power microcontroller with 2KB of flash, 256 bytes of RAM running at 16MHz. This processor is responsible for filtering the inputs and running a software Phase Loop Lock (PLL) to generate a stable output for the main FireFly3 board to use as a reference (see [1] for more details).

We now evaluate the sensitivity and timing stability of the new circuit. In [16] the authors measure and model the magnetic fields produced by various power line configurations to evaluate the potential health hazard on humans. Their measurements show that the magnetic field strength near overhead transmission lines can be as strong as 17 milli-gauss and drops down to a still detectable 3-4 milli-gauss at 60 meters away. This indicates that in most outdoor urban environments it is still possible to detect these magnetic fields in and around buildings. In [17], the authors measure typical magnetic field values in homes ranging from .001 gauss to 10 gauss near appliances and as high as 100 gauss in industrial settings. By contrast, the earth's natural magnetic field at 60Hz is much weaker $2 \cdot 10^{-7}$ gauss making these artificial signals relatively simple to detect. Figure 9 shows our experimental setup where the sensor node is placed at a fixed distance of 1m from a 12 gauge wire that is connected to a 120VAC power line. Figure 10 shows that the received signal intensity linearly increases with antenna length. This is to be expected with Hertzian antenna configurations where the dipole element is significantly smaller than the target wavelength. The usable signal noise-floor for detection is about $2mV$. Next, we take a receiver with a short (10 cm) antenna and adjust the distance of the receiver from the source wire. Figure 11 shows the signal drop-off with respect to distance. The small antenna decreases sensitivity allowing us to more clearly see the trend. In practice, an antenna of length 10-15cm provides relatively good coverage in most building environments.

Next, we measure the jitter associated with the 60Hz signal timing from the electric field sensor as compared to the original magnetic field design. The frequency of an AC

power line typically has a stability of about $5 \cdot 10^5$ [18]. In the past, devices like alarm clocks and home appliances have used a direct connection to the power-line as a source for keeping wall-clock time. This local stability is on the order of a relatively poor quality clock crystal. However, in order for power to be delivered efficiently across the country, the phase difference between any two points should remain fairly constant. [18] shows the differential delay to have a stability of 1 part in 10^8 over a 24 hour period. Hence, the power grid is phase-coherent. In the United States, there are four main power grids that cover the entire country. Most buildings are supplied with multiple phases of power. This in combination with the orientation between our receiver and nearby dominant magnetic fields, the detected signal will lead or lag with respect to the original signal resulting in a a phase offset. This means that our receivers achieve *syntonization* with each other. Syntonization is defined as when two clocks are frequency locked, but they may have a phase offset (hence we call our receiver a *Syntonistor*). Since the power lines act as a global broadcast, even if the frequency shifts, each node in the network still receives a common global clock tick. This means that after initialization, all clock rates are identical and do not drift. This initialization step requires the local exchange of messages using the protocol described in [1]. This protocol is based on the Flooding Time Synchronization protocol [2] and is only required at startup or during an error recovery operation. Figure 12 shows a histogram of pairwise jitter between two nodes collected over four hours. We clearly see that the jitter captured by the electric front-end is significantly less than the original magnetic receiver. This both simplifies the PLL design and inevitably generates filtered edges with lower total amounts of jitter. We attribute this improvement to the increased stability of the electric field since it is less influenced by changing current draw and the fact that the front-end has fewer components on the signal path.

B. Error Recovery

The Syntonistor has the ability to detect when the timing of the incoming signal unexpectedly increases beyond the normal jitter threshold. This could happen for various reasons including a physical change in the environment, a new nearby appliance getting powered up, or even a power outage. In response to these sorts of errors, the Syntonistor will raise its error line which signals to the main sensor node that the PPS value may no longer be accurate. If the error line remains high for a long enough period (based on local clock drift), the node must fall back to an existing software synchronization technique. For example, the node can periodically pass messages with a neighbor to update its clock. Once the power-line signal stabalizes again, the error line from the Syntonistor will transition from high to low. At this point, the node will send a message back to the master node requesting a new set of flooding time synchronization messages. This high to low transition naturally happens the first time a node is powered on. One possible optimization is for nodes to only request the phase offset based on their neighbors. In practice this works well, however over time this could result in nodes drifting with respect to the master if groups of nodes go in and out of synchronization in lock-step.

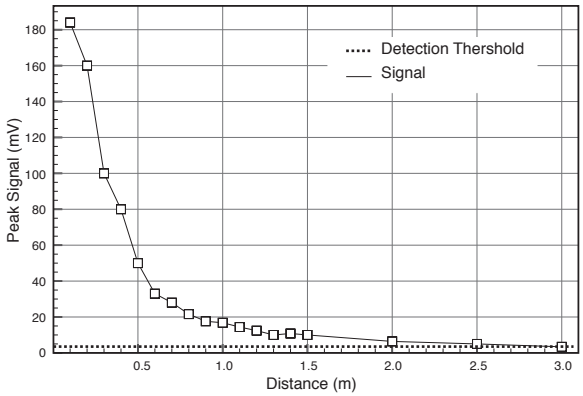

Fig. 11. FET peak signal strength vs distance given 10cm antenna.

C. Hardware Clock Rate Adjustment

Most time synchronization techniques rely on rate adjustment to slowly change the system's notion of time without introducing logical discontinuities that might occur if time

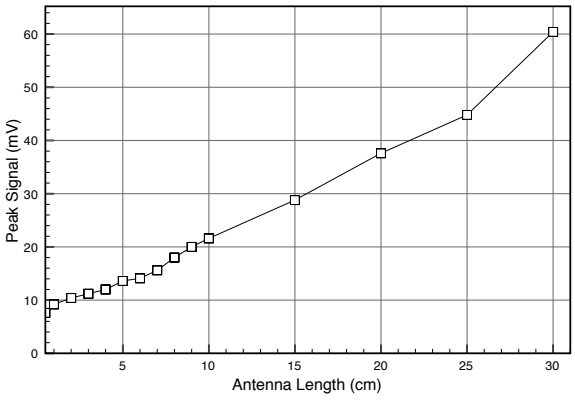

Fig. 10. FET antenna length (orthogonal to cable) vs signal with node positioned 1m from cable.

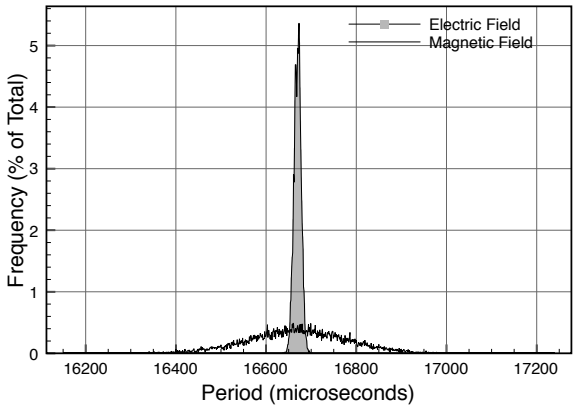

Fig. 12. Jitter in electric field compared to magnetic field (original Syntonistor) front-end

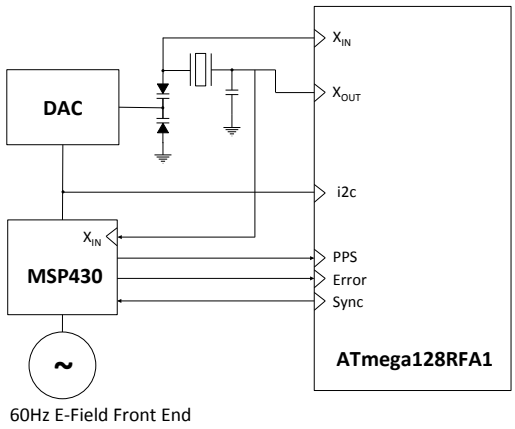

Fig. 13. Hardware Clock Rate Adjustment Block Diagram

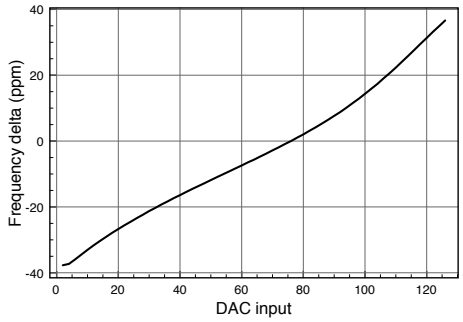

Fig. 14. Crystal frequency change vs DAC input

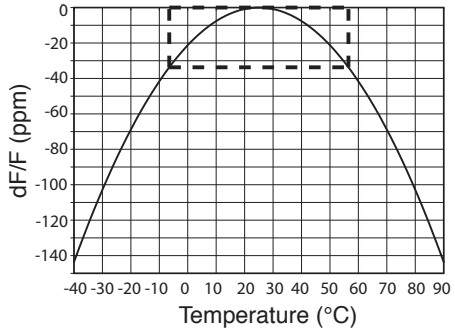

Fig. 15. Sustainable operating region

suddenly changed. This can be accomplished in both software and hardware. On the FireFly3, we provide a simple circuit that allows both the main FireFly3 processor as well as external devices to tune the low-power 32KHz crystal used for long-term time keeping.

Oscillator circuits contain a crystal, load capacitors and a feedback loop to generate a periodic electrical signal that is used as a clock input to processors. Typically the feedback loop is internal to the MCU while the crystal and load capacitors are connected externally (load capacitances are typically specified by the crystal manufacturer). Depending on the load capacitance applied to the crystal, it is possible to *push* or *pull* the crystal frequency shifting it slightly up or down. This property is often utilized in high-speed digital tuning circuits to adjust clock frequencies. In our design, we introduce a variable voltage controlled capacitor (varactor) along with an ultra-low-power digital-to-analog converter (DAC) to allow runtime tuning of the crystal hardware. Unlike most external rate adjustable clock circuits, this approach can be integrated within the normal feedback control-path of the oscillator without requiring external buffering chips. This significantly reduces both the cost and power requirements of the circuit.

Figure 13 shows a block diagram of how the Syntonistor interfaces with the FireFly3's MCU. The physical hardware for the varactor and DAC are located on the core FireFly3 module. The MSP430 and the electric field front-end are co-located on the syntonistor add-on board. The FireFly3 provides a dedicated port allowing the syntonistor to access the control lines for the DAC enabling it to adjust the host processor's clock rate. The system is designed such that the FireFly node will operate normally with a fixed crystal frequency if the Syntonistor is not present. The input to the varactor is provided by the MCP40D18 DAC from Microchip, which provides 7-bit resolution and consumes $1uW$ of power when idle. In order to evaluate the degree to which we are able to push and pull the clock, we use a test program on the FireFly3 node that outputs a square wave at a known frequency (in this case 100Hz) using the 32KHz clock. We capture 1000 period values for each DAC input setting using a logic analyzer connected to a PC[19]. Figure 14 shows the average frequency change with respect to the DAC input. Values of 0 and 127 at the DAC's extremes cause the clock to fail and hence do not have a corresponding measurement. One can clearly see that the circuit has the ability to adjust the crystal's frequency by more than 75 ppm. Figure 15 shows how this range maps to the crystal's temperature-to-frequency relationship based on the manufacturer's data sheet. It is worth noting that the circuit has the ability to temperature compensate the clock over a range of -8 to 58 degrees Celsius. Since the ATmega128RFA1 has an internal temperature sensor, it is possible to perform temperature-based clock adjustment even without the syntonistor hardware since the DAC and varactor are part of the main FireFly3 node.

The MSP430 on the syntonistor internally generates a Pulse Per Second (PPS) filtered signal based on the 60Hz electric field front-end. We can use this as a reference signal to trim the clock on the FireFly3 assuming the MSP430 is also able to read the FireFly3's clock. Since we ideally want the circuit to adjust the FireFly3 clock even while the FireFly3 is asleep, this requires a hardware interconnect. By carefully placing the varactor on the input side of the crystal feedback circuit (where it can exert more of an impact on frequency) it is possible to tap the driven output line of the ATmega128RFA1 processor and feed this into the low-frequency clock input of the MSP430. The output channel of the MSP430 can then be connected through a large resistor to ground. This configuration allows the MSP430's asynchronous low-power clock register to accumulate ticks directly from the FireFly3's clock. The MSP430 can then run a simple control loop that counts the number of expected 32KHz clock ticks within each PPS signal synthesized from the 60Hz power line. For example, within one second, the MSP430 expects to see 32,768 clock ticks from the FireFly3 node. Collected over a long enough period of time,

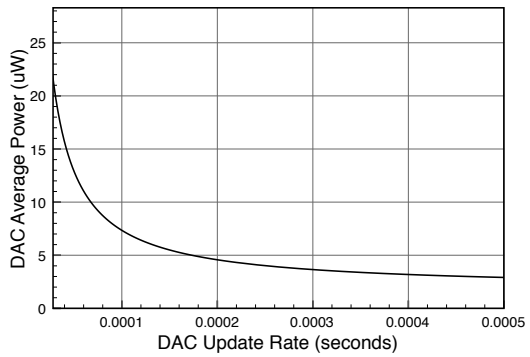

Fig. 16. DAC power vs sampling rate

Component	Typical Power (μW)	Max Power (μW)
NE3503M04	0.5	10.0
OPA2369	0.84	1.0
OPA333	10	30
MCP4018	1.1	22
MSP430G2452	5	19.8
Total	**17.44**	**82.8**

TABLE I. THIS TABLE SHOWS A BREAKDOWN OF AVERAGE POWER CONSUMED BY THE MAIN HARDWARE COMPONENTS IN THE SYNTONISTOR AND CLOCK ADJUSTMENT CIRCUIT.

the drift between the two clock sources becomes apparent. For example, if the syntonistor counts 327,685 counts over a 10 second period (referenced from the 60Hz signal), then it can compute that the FireFly3 clock is running too quickly (we expect 327,680) and needs to be slowed down. We implement a simple binary threshold control law on the syntonistor to track these values. The firmware image for the MSP430 consumes 1540 bytes of flash and 176 bytes of RAM.

Next, we evaluate the energy impact of the new Syntonistor circuit design. The previous magnetic field sensing system consumed on average $58uW$. Our new design removes the INA front-end and replaces it with a lower-power op-amp that saves around $30uW$. We also have an additional DAC to control the crystal's rate. The updated components and their associated power consumptions are listed in Table I. The DAC consumes $1.8uW$ of power when idle and up to $22uW$ of power when active. The DAC requires $22uS$ to change state. Depending on how often updates are made to the crystal's rate, the overall power of the DAC is shown in Figure 16. The static consumption of the MSP430 processor when running with the analog front-end components was measured to be about $15uW$. Including the DAC, the entire system consumes on average about $17.4uW$ which is 60% less then the original design. Since the electric field does not need to resonate, it can startup within micro-seconds instead of hundreds of milliseconds as in the previous design. This makes it conceivable that the entire front-end circuit could be duty-cycled to wakeup just before it expects one of the 60Hz edges. In the future, this optimization could further reduce power consumption.

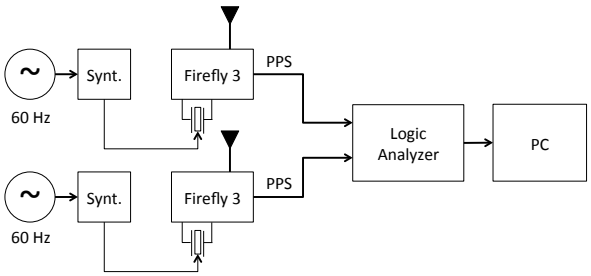

Fig. 17. Pairwise performance experimental setup

V. SYSTEM EVALUATION

We now evaluate the performance of the FireFly3 platform in terms of drift-rate, asynchronous sampling-error and sleep-energy.

In order to accurately monitor drift over an extended period of time, we utilized a variant of our original experimental setup used to measure the push-and-pull ability of our hardware clock. Figure 17 shows two FireFly3 nodes with GPIO output lines directly connected to a logic analyzer that can be monitored by a PC. The GPIO pins are set to toggle based on an interrupt driven by the 32KHz low-power clock. This is the clock typically used on sensor nodes by the operating systems and event schedulers since it operates when the node is in deep sleep. The 32KHz clock is used to generate a 10 second square wave. The Syntonistors then independently adjust the rate of the 32KHz clock based on the 60Hz power line signal. In this experiment, we do not use the PPS signal from the Syntonistor to adjust the FireFly's time. Instead we rely exclusively on the rate adjustment circuit to keep them at similar rates. Using the logic analyzer we can precisely measure the jitter between each rising edge to within 40 nanoseconds (the logic analyzer samples at 25MHz). Over a period of 24 hours we see that without the Syntonistor, the nodes drift by 1.2 seconds which corresponds to a crystal precision of 14ppm. With the Syntonistor active, the two nodes drift by only 864 microseconds which corresponds to a 0.01ppm clock. In this example, both nodes are within close physical proximity to one another. However in practice, one node might be far enough away that it will be exposed to different temperature or acceleration environments, which will significantly impact the non-corrected drift, but will have little impact on the corrected version.

Next, we compare the energy difference between three different approaches for synchronization using the Syntonistor. The first is a "Fixed periodic" approach where the main processor wakes up every second to resynchronize with the PPS output of the Syntonistor (this was the original scheme from [1]). We then propose an enhanced version of this scheme where the host processor conservatively estimates its drift with respect to the Syntonistor PPS output and schedules itself to wake up only near boundaries when it could lose track of the PPS count. For example, if the host processor is drifting at a rate of 50ppm with respect to the Syntonistor clock, it can safely sleep for 2.5 hours without worrying about drifting by more than a single PPS timer tick. For extended periods of deep sleep, the system can use a drift and resynchronize approach to keeping track of time. We call this approach "rate adaptive"

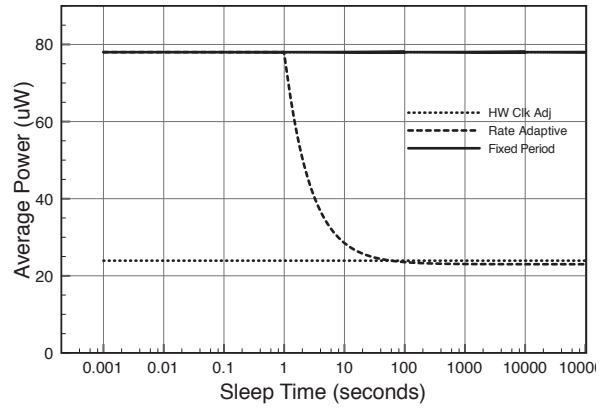

Fig. 18. Impact of rate adjustment on energy

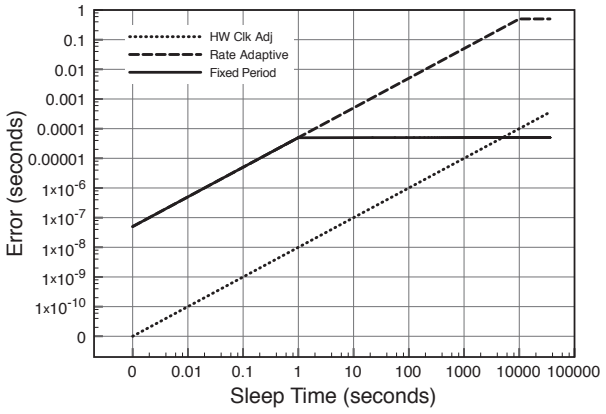

Fig. 19. Asynchronous sample time error vs sleep interval

Syntonistor tracking. The general idea is that the processor must wakeup to sample an event, set an interrupt for the next PPS edge, go to sleep to conserve energy, wakeup just before the edge and adjust its clock. Our final approach uses hardware clock rate adjustment where the CPU can essentially just sleep for extended periods.

In our first comparison, we investigate the impact of each approach on total energy overhead while sleeping for an extended period of time. With the fixed periodic approach, the system has to frequently wake up to resynchronize. Even if this is connected with an interrupt routine triggered by the PPS clock, it still requires 5-10ms for the main CPU to wake up and resynchronize. In the more intelligent rate adaptive approach, the wake up overhead becomes less significant as the sleep period increases. Waking up once every 2.5 hours maintains an extremely low duty-cycle. We will later see that this has a negative impact on asynchronous timestamping of events. With the hardware clock rate adjusted approach, the energy consumption is always relatively static. Figure 18 shows an analytical comparison of the average power required to maintain (the most accurate level of clock adjustment) consumed by the three approaches as the desired sleep time increases. We see that beyond about 30 seconds the rate adaptive approach and the hardware approach are essentially equivalent. Below 30 seconds, the rate adaptive approach must frequently synchronize until it eventually becomes equivalent to the fixed periodic approach at small time scales. Below a 1 second sleep interval, the system wakes up once per second periodically to maintain accuracy which is why the power remains constant. Both rate adaptive and hardware-assisted adjustment significantly out perform the fixed periodic approach at longer time scales. In these evaluations, we assume 50ppm clock crystals that must remain synchronized to within 1 second of the Syntonistor's PPS output so as to stay in phase with the signal.

Next, we evaluate the ability for the system to accurately timestamp asynchronous events while the processor has been in deep-sleep for an extended period of time. If an event occurs while the CPU is sleeping, it must be timestamped based on the best available information after wake up. Figure 19 shows the drift error with respect to sleep interval under different approaches. Note, that both axes are log-scale. In the fixed periodic update approach, the node would have synchronized in the worst-case one second in the past, which is why we see it perform the best over long sleep periods. This is of course at the cost of expending significantly more energy. The hardware clock rate adjustment approach is linear based on its 0.01ppm clock drift rate. Initially, this outperforms the periodic approach since its internal clock is 50ppm, but eventually the periodic resynchronization accounts for this drift and outperforms the hardware clock rate scheme at about 1.6 hours. The rate adaptive approach suffers the most since it could have drifted by up to half a PPS period over an extended period of time before it resynchronizes. The plateau at 2.7 hours is the result of the resynchronization interval used to catch drift. We could imagine combining rate adaptation with the hardware clock rate adjustment and earlier wake up periods in a manner where designers could tune accuracy and energy to better meet application requirements.

VI. Limitations

The main limitation to this approach is that additional hardware adds to both the cost and energy consumption. For example, if one were to take the improved electric field front-end and remove the DAC used for clock rate adjustment the energy would be further reduced. For extended sleep periods the software rate adaptive approach could provide similar performance. It would also be possible to back-compute timestamp arrivals based on future synchronization points to overcome the poor asynchronous timestamping error. However, this would result in increased latency. In general, the cost overhead of adding the additional hardware is minimal and it provides an elegant solution that outperforms software-based approaches in certain scenarios and simplifies overall software design. We believe for a real-time wireless sensor networking platform, both of these hardware components could play significant roles in building robust systems in the future.

Another limitation of using a global broadcast reference for synchronization is that the system is only internally synchronized. This means that external devices will have their own (different) notion of time. Since the power line is not particularly stable over short time periods (differential stability is high, but not necessarily local stability), this means that external devices need to periodically communicate with the infrastructure to maintain time synchronization.

VII. Conclusion and Future Work

Accurate timing is an important component for distributed systems that are designed to provide high levels of QoS support. In wireless sensing systems this can be used to improve fine-grained event ordering and communication scheduling. In this paper, we presented the FireFly3 WSN platform that improves timing through use of an external synchronization module and a hardware clock-rate adjustment circuit. We present an improved circuit design that is able to synchronize with the 60Hz electromagnetic fields emitted from power lines. This signal can be used as a global clock source for the network. We then introduced a novel hardware clock rate adjustment system that allows for low-power clock rate adjustment of the main FireFly3's processor externally even while it is in a deep-sleep mode. This enables more precise instantaneous timestamping of asynchronous events when the node is in deep-sleep and saves energy that would otherwise have been wasted on periodic wake up overhead. Even without the Syntonistor, the hardware clock rate adjustment can be used in conjunction with the main processor's temperature sensor to perform temperature-based clock rate adjustment. We show that our new platform has a clock stability of 0.01ppm and consumes on average $22uW$ in deep-sleep while maintaining synchronization. In the future we would like to investigate the potential for new kinds of low-power MAC protocols that capitalize on out-of-band drift-rate adjustment. We also believe that the average power can be further reduced by duty-cycling how the Syntonistor samples the sensor front-end.

Acknowledgment

We would like to thank Ge Yang for his work on the MSP430 PLL software and Thomas Schmid for his valuable discussions related to this work.

References

[1] A. Rowe, V. Gupta, and R. R. Rajkumar, "Low-power clock synchronization using electromagnetic energy radiating from ac power lines," in *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, SenSys '09, (New York, NY, USA), pp. 211–224, ACM, 2009.

[2] M. Maroti and B. Kusy and G. Simon and A. Ledeczi, "The flooding time synchronization protocol," *Proc. ACM Sensys*, 2004.

[3] J. Elson and L. Girod and D. Estrin, "Fine-grained network time synchronization using reference broadcast," *USENIX OSDI*, 2002.

[4] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, "Efficient network flooding and time synchronization with Glossy," in *Proceedings of the 10th IEEE International Conference on Information Processing in Sensor Networks*, IPSN '11, pp. 73–84, Apr. 2011.

[5] L. Lamport, "Time, clocks and ordering of events in distributed systems," *Communications of the ACM*, 1978.

[6] D. Mills, "Internet time synchronization: The network time protocol," *IEEE Transactions on Communications*, 1991.

[7] Z. S. Gusell, R., "The accuracy of clock synchonization achieved by tempo," *IEEE transactions on Software Engineering*, 1989.

[8] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, 1989.

[9] O. W. Kopetz, H., "Clock synchonization in distributed real-time systems," *IEEE Computer*, August 1987.

[10] S. Ganeriwal and R. Kumar and M. B. Srivastava, "Timing-sync protocol for sensor networks," *Proc. ACM Sensys*, 2003.

[11] T. Schmid, P. Dutta, and M. B. Srivastava, "High-resolution, low-power time synchronization an oxymoron no more," in *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '10, (New York, NY, USA), pp. 151–161, ACM, 2010.

[12] G. Werner-Allen, G. Tewari, A. Patel, M. Welsh, and R. Nagpal, "Firefly-inspired sensor network synchronicity with realistic radio effects," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pp. 142–153, 2005.

[13] C. Z. S. R. S. M. Schmid, T., "Temperature compensated time synchronization," in *IEEE Embedded Systems Letters*, 2009.

[14] Rowe A., Mangharam R., and Rajkumar R., "RT-Link: A Time-Synchronized Link Protocol for Energy-Constrained Multi-hop Wireless Networks," *SECON*, 2006.

[15] F. Schmuck and F. Cristian, "Continuous clock amortization need not affect the precision of a clock synchronization algorithm," in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, (New York, NY, USA), pp. 133–143, ACM, 1990.

[16] Olsen, R. G., Deno, D., Baishiki, R. S. , "Magnetic fields from electric power lines theory and comparison to measurements," *IEEE Transactions on Power Delivery*, 1988.

[17] Deno, D. , "Sources and Structures of Magnetic and Electric Fields in the home," *23rd Hanford Life Science Symposium*, 1984.

[18] M. H. Allan, D., "Time transfer using nearly simultaneous reception times of a common transmission," *26th Annual Symposium on Frequency Contro*, pp. 309–316, 1972.

[19] "http://www.saleae.com/logic,"