Time Synchronization in Heterogeneous Sensor Networks

Isaac Amundson[1], Branislav Kusy[2], Peter Volgyesi[1], Xenofon Koutsoukos[1], and Akos Ledeczi[1]

[1] Institute for Software Integrated Systems (ISIS)
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235, USA
Email: isaac.amundson@vanderbilt.edu
[2] Department of Computer Science
Stanford University
Stanford, CA 94305, USA

Abstract. Time synchronization is a critical component in many wireless sensor network applications. Although several synchronization protocols have recently been developed, they tend to break down when implemented on networks of heterogeneous devices consisting of different hardware components and operating systems, and communicate over different network media. In this paper, we present a methodology for time synchronization in heterogeneous sensor networks (HSNs). This includes synchronization between mote and PC networks, a communication pathway that is often used in sensor networks, but has received little attention with respect to time synchronization. In addition, we evaluate clock skew compensation methods including linear regression, exponential averaging, and phase-locked loops. Our HSN synchronization methodology has been implemented as a network service and tested on an experimental testbed. We show that a 6-hop heterogeneous sensor network can be synchronized with an accuracy on the order of microseconds.

1 Introduction

Wireless sensor networks (WSNs) consist of large numbers of cooperating sensor nodes that can be deployed in practically any environment, and have already demonstrated their utility in a wide range of applications including environmental monitoring, transportation, and healthcare. These types of applications typically involve the observation of some physical phenomenon through periodic sampling of the environment. In order to make sense of the individually collected samples, nodes usually pass their sensor data through the network to a centralized *sensor-fusion* node where it can be combined and analyzed. We call this process *reactive data fusion*.

One important aspect of reactive data fusion is the need for a common notion of time among participating nodes. For example, acoustic localization requires the cooperation of several nodes in estimating the position of the sound source

based on time-of-arrival data of the wave front [1], [2]. This may require up to 100-microsecond accurate synchronization across the network. Another example is acoustic emissions (AE), the stress waves produced by the sudden internal stress redistribution of materials caused by crack initiation and growth [3]. As the speed of sound is typically an order of magnitude higher in metals than in the air, the synchronization accuracy required for AE source localization can be in the tens of microseconds.

Accurately synchronizing the clocks of all sensor nodes within a heterogeneous network is not a trivial task. Existing WSN time synchronization protocols (e.g. [4], [5], [6], [7], [8]) perform well on the devices for which they were designed. However, these protocols tend to break down when applied to a network of *heterogeneous* devices. For example, the Routing-Integrated Time Synchronization (RITS) protocol [8] was designed to run on the Berkeley motes, and assumes the operating system is tightly integrated with the radio stack. Attempting to run RITS on an 802.11 network can introduce an unacceptable amount of synchronization error because it requires low-level interaction with the hardware, which is difficult to attain in PCs. Reference Broadcast Synchronization (RBS) [5], although portable when it comes to operating system and computing platform, is accurate only when all nodes have access to a common network medium. A combined network of Berkeley motes and PDAs with 802.11b wireless network cards, for example, would be difficult to synchronize using RBS alone because they communicate over different wireless channels. In addition, the communication pathway between a mote and PC is realized using a serial connection. Synchronization across this interface is essential when attempting to combine time-dependent sensor data from each device. Furthermore, it may be desirable to have several mote-PC gateways, since often it is not always possible to extract data fast enough through a single base station. In large networks, this also enables data packets to reach the base station in a fewer number of hops, thus minimizing delay and conserving energy. Although time synchronization across this interface has previously been explored (see Section 2), it has not been implemented in software.

Our work focuses on achieving microsecond-accuracy synchronization in heterogeneous sensor networks (HSNs). HSNs are a promising direction for developing large sensor networks for a diverse set of applications [9], [10], [11]. We consider a multi-hop network consisting of Berkeley motes and Linux PCs, a dominant configuration for reactive data fusion applications. Mote networks consist of resource-constrained devices capable of monitoring environmental phenomena. PCs can support higher-bandwidth sensors such as cameras, and can run additional processing algorithms on the collected data before routing it to the sensor-fusion node. In this sense, we model both motes and PCs as sensor nodes. Time synchronization in both mote and PC networks have been studied independently, however, a sub-millisecond software method to synchronize these two networks has not yet been developed to the best of our knowledge.

In this paper, we present a methodology for HSN time synchronization that utilizes a combination of existing synchronization protocols. Our methodology supports reactive data fusion, incurs little overhead of network resources, and

has synchronization error on the order of microseconds. In addition, we have implemented a time synchronization service for reactive data fusion applications, which allows the application developer to focus on aspects of sensor fusion, and not the underlying aggregation mechanism. To achieve accurate cross-platform synchronization, we have developed a technique for synchronization between a mote and PC that is implemented completely in software, and remains effective when multiple mote-PC connections exist within the network.

The rest of this paper is organized as follows. Section 2 describes existing synchronization protocols for WSNs. We present the problem of HSN time synchronization in Section 3. We discuss the sources of synchronization error in Section 4, and clock skew compensation in Section 5. In Section 6, we present our methodology and implementation for HSN time synchronization, and in Section 7 our evaluation results. Section 8 concludes.

2 Related Work

Synchronization protocols can be classified as *sender-receiver*, in which one node synchronizes with another, or *receiver-receiver*, in which multiple nodes synchronize to a common event. Both have their advantages, and each can provide synchronization accuracy on the order of microseconds using certain configurations [12]. An in-depth survey on time synchronization in WSNs can be found in [12].

Several sender-receiver synchronization protocols have been developed for the Berkeley motes and similar small-scale devices that provide microsecond accuracy. Elapsed Time on Arrival (ETA) [13] provides a set of application programming interfaces for an abstract time synchronization service. In ETA, sender-side synchronization error is essentially eliminated by taking the timestamp and inserting it into the message *after* the message has already begun transmission. On the receiver side, a timestamp is taken upon message reception, and the difference between these two timestamps estimate the clock offset between the two nodes. RITS [8] is an extension of ETA over multiple hops. It incorporates a set of routing services to efficiently pass sensor data to a network sink node for data fusion.

In [14], mote-PC synchronization was achieved by connecting the GPIO ports of a mote and IPAQ PDA. The PDA timestamped the output of a signal, which was captured and timestamped by the mote. The mote then sent the timestamp back to the PC, which was able to calculate the clock offset between the two. Although using this technique can achieve microsecond-accurate synchronization, it was implemented as a hardware modification rather than in software.

Reference Broadcast Synchronization (RBS) [5] is a receiver-receiver protocol that minimizes error by taking advantage of the broadcast channel found in most networks. Messages broadcast at the physical layer will arrive at a set of receivers within a tight time bound due to the almost negligible propagation time of sending an electromagnetic signal through air. Nodes then synchronize their clocks to the arrival time of the broadcast message.

3 Problem Statement

Our goal is to provide accurate time synchronization to reactive data fusion applications in HSNs. We refer to the combination of these components as a *config-*

uration. Our testbed configuration consists of Mica2 motes and Linux PCs. There are two physical networks, the B-MAC network [15] formed by the motes and the 802.11 network containing the PCs. The link between the two is achieved by connecting a mote to a PC using a serial connection. This mote-PC configuration is chosen because it is representative of HSNs containing resource-constrained sensor nodes for monitoring the environment and resource-intensive PCs used for high-bandwidth sensing and computation.

Ideally, a single synchronization methodology would suffice for the entire HSN. However, no protocol has been developed that can achieve this. Instead, we turn to the underlying methodologies found in existing protocols, and use them in conjunction. Individually, the mote and PC networks have been studied extensively. However, the interaction between the two has not been sufficiently investigated. To understand how synchronization affects these connections, we first adopt a system model for time synchronization.

System Model Each sensor node in a WSN is a computing device that maintains its own local clock. Internally, the clock is a piece of circuitry that counts oscillations of a quartz crystal, energized at a specific frequency. When a certain number of these oscillations occur, a *clock-tick* counter is incremented. This counter is accessible from the operating system and its accuracy (with respect to atomic time) depends on the quality of the crystal, as well as various operating conditions such as temperature, pressure, humidity, and supply voltage. When a sensor node registers an event of interest, it will access the clock-tick counter and record a *timestamp* reflecting the time at which the event occurred.

Some protocols synchronize nodes to atomic time, often referred to as *real-time* or *Coordinated Universal Time* (UTC). Irrespective of whether a given protocol synchronizes to UTC, it is often convenient to represent the occurrence of an event according to some universal time standard. We use the notation t to represent an arbitrary UTC time, and the notation t_e to represent the UTC time at which an arbitrary event e occurred. Because each node records a timestamp according to its own clock, we specify the local time on node N_i at which event e was detected by the timestamp $N_i(t_e)$.

Although the two timestamps $N_i(t_e)$ and $N_j(t_e)$ correspond to the same real-time instant t_e, this does not imply that $N_i(t_e) = N_j(t_e)$; the clock-tick counter on node N_i may be offset from N_j. Therefore, from the perspective of node N_i, we define the *clock offset* with N_j at real-time t as $\phi_j^i(t) = N_i(t) - N_j(t)$. It may be the case that the offset changes over time. In other words, the *clock rate* of node N_i, $\frac{dN_i(t)}{dt}$, may not equal the ideal rate ($\frac{dN_i(t)}{dt} = 1$). We define the ratio of clock rates of two nodes as the *relative rate*, $rr_j^i = \frac{dN_i(t)}{dN_j(t)}$. The relative rate is a quantity directly related to the *clock skew*, defined as the difference between clock rates, and is used in our clock skew compensation methods. We refer to clock offset and clock rate characteristics as a node's *timescale*.

Practically all synchronization protocols can be implemented using *timescale transformation*. Rather than setting one clock to another, clocks advance uninhibited, and instead a reference table is maintained that keeps track of the clock offsets and clock drift between a node and its neighbors. The reference table is

used to transform a clock value from one timescale to another, providing each node with a common notion of time. Clock adjustment is disadvantageous in WSNs because it leads to increased overhead and possible loss of monotonicity [16]. Therefore, in subsequent sections, we discuss synchronization solely from the perspective of timescale transformation.

4 Sources of Synchronization Error

Synchronization requires passing timestamped messages between nodes. However, this communication has associated message delay, which has both deterministic and nondeterministic components that introduce error into the timescale transformation. We call the sequence of steps involved in communication between a pair of nodes the *critical path*. Figure 1 illustrates the critical path in a wireless connection. The critical path is not identical for all configurations, however, it can typically be characterized by the steps outlined in the figure (for more details, see for example [5], [6], [7]).

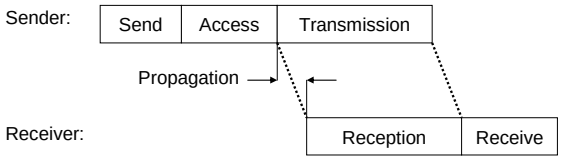

Fig. 1. Critical path

In both the mote and PC networks, the Send and Receive times are the delays incurred as the message is passed between the application and the MAC layer. These segments are mostly nondeterministic due to system call overhead and kernel processing. The Access time is the least deterministic segment in the critical path. It is the delay that occurs in the MAC layer while waiting for access to the communication channel. The Transmission and Reception times are the delays incurred from transmitting and receiving a message over the physical layer, bit by bit. They are mostly deterministic and depend on bit rate and message length. The Propagation time is the time the message takes to travel between the sender and receiver. Propagation delay is highly deterministic.

For the mote-PC serial pathway, the Send and Receive times are similar to wireless communication, however, the Access time is nonexistent. Because the mote-PC connection does not have flow control, pending messages are immediately transmitted without having to wait for an available channel. The Transmission time starts when the data on the sender is moved in 16-byte chunks to the buffer on the UART, and ends after the data is transmitted bit-by-bit across the serial port to the receiver. Similar to wireless networks, the Propagation time is minimal. The UART on the receiver places the received bits into its own 16-byte buffer. When the buffer is almost full, or a timeout occurs, it sends an interrupt to the CPU, which notifies the serial driver, and the data is transferred to main memory where it is packaged and forwarded to the user-space application.

In addition to the error from message delay nondeterminism, synchronization accuracy is also affected by clock skew when a pair of nodes operate for extended periods of time without correcting their offset. For example, suppose at real-time t, nodes N_1 and N_2 exchange their current clock values at local times $N_1(t)$ and $N_2(t)$, respectively. At some later time, an event e occurs that is detected and timestamped by N_1, which sends its timestamp $N_1(t_e)$ to N_2. If the clock rates on each node were equal, N_2 would simply be able to take the previously calculated offset and use it to transform the event timestamp $N_1(t_e)$ to the corresponding local time $N_2(t_e) = N_1(t_e) + \phi_2^1(t)$. However, if the relative rate $rr_2^1(t)$ is not equal to 1, but $1 + 20$ ppm[3], for example, an attempt to convert $N_1(t_e)$ to the local timescale would result in an error of $20 * 10^{-6} * (N_2(t_e) - N_2(t))\mu s$. If the interval between the last synchronization and the event was one minute, the resulting error due to clock skew alone would amount to 1.2 milliseconds!

For accurate synchronization, it is therefore necessary to minimize the nondeterministic sources of error in the critical path, and account for the deterministic sources by appropriately adjusting the timescale transformation. Clock skew compensation is necessary for minimizing synchronization error when nodes run for long periods of time without updating their offset. With an estimation of clock offset and relative rate, a complete timescale transformation, which converts an event timestamp $N_j(t_e)$ from the timescale of node N_j to the timescale of N_i, can be defined as

$$N_i(t_e) = N_i(t_s) + rr_j^i(t_s)[(N_j(t_e) - N_j(t_s)]$$

where $N_i(t_s)$ and $N_j(t_s)$ are the respective local times at which nodes N_i and N_j exchanged their most recent synchronization message, s.

5 Clock Skew Compensation

Independent of synchronization protocol, there are several options for clock skew compensation. The simplest is to do nothing. In some applications, event detection and synchronization always occur so close together in time that clock skew compensation is unnecessary. However, when it does become necessary, nodes must exchange timestamps periodically to ensure their clocks do not drift too far apart. In resource-constrained WSNs, this may be undesirable because it can result in high message overhead. To keep message overhead at a minimum, nodes can exchange synchronization messages less frequently and instead maintain a history of their neighbors' timestamps. Statistical techniques can then be used to produce an accurate estimate of clock offset at any time instant.

A linear regression fits a line to a set of data points such that the square of the error between the line and each data point is minimized overall. By maintaining a history of n local-remote timestamp pairs, node N_i can derive a linear relation $N_i(t) = \alpha + \beta N_j(t)$ and solve for the coefficients α and β. Here, β represents an estimation of $rr_j^i(t)$. A problem arises when attempting to improve the quality of the regression by increasing the number of data points. This can result in high

[3] Parts per million (10^{-6}). A relative rate of 1 ppm means that one clock ticks $1\mu s/s$ faster than the other.

memory overhead, especially in dense networks. However, it has been shown that sub-microsecond clock skew error can be achieved with as few as six timestamps in mote networks [7].

Exponential averaging solves the problem of high memory overhead by keeping track of only the current relative rate and the most recent neighbor-local synchronization timestamp pair. When a new timestamp pair is available, the relative rate is adjusted. Because the relative rate estimate is partially derived from its previous value, there will be a longer convergence time before an accurate estimate is reached. This can be reduced by providing the algorithm with an initial relative rate, determined experimentally.

The phase-locked loop (PLL) is a mechanism for clock skew compensation used in NTP [17]. The PLL compares the ratio of a current local-remote timestamp pair with the current estimate of relative rate. The PLL then adjusts the estimate by the sum of a value proportional to the difference and a value proportional to the integral of the difference. PLLs generally have a longer convergence time than linear regression and exponential averaging, but have low memory overhead. A diagram of a PLL implementation is illustrated in Figure 2. The Phase Detector calculates the relative rate between two nodes and compares this with the output of the PLL, which is the previous estimate of the relative rate. The difference between these two values is the phase error, which is passed to the second-order Digital Loop Filter. Because we expect there to be some amount of phase error, we choose a filter with an integrator, which allows the PLL to eliminate steady-state phase error. To implement this behavior in software, a digital accumulator is used, and is represented by $y(t) = (K_1 + K_2)u(t) - 10K_2K_1u(t-1) + 10K_2y(t-1)$. The resulting static phase error is passed to the Digitally Controlled Oscillator (DCO). The DCO sums the previous phase error with the previous output, which produces the current estimate of relative clock rate, and is fed back into the PLL. Techniques for selecting the gains are presented in [17].

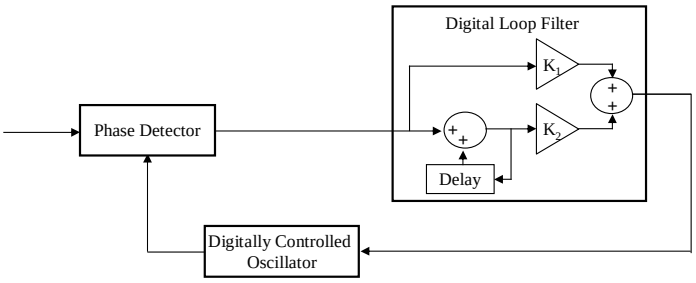

Fig. 2. Phase-locked loop

6 Time Synchronization in HSNs

In this section, we present our HSN synchronization methodology and the architecture of our synchronization service.

Synchronization Methodology The accuracy of a receiver-receiver synchronization protocol (such as RBS) is comparable to sender-receiver synchronization (such as RITS) in mote networks. However, receiver-receiver synchronization has greater associated communication overhead, which can shorten the lifetime of the network. Therefore, we selected RITS to synchronize the mote network in our HSN.

Synchronization of PC networks has been studied extensively over the past four decades, however, popular sender-receiver protocols such as the Network Time Protocol (NTP) [18] only provide millisecond accuracy. This is acceptable because PC users typically do not require greater synchronization precision for their applications. For microsecond-level synchronization accuracy in PC networks, a receiver-receiver protocol such as RBS outperforms sender-receiver protocols because it has less associated message delay nondeterminism. We therefore use RBS to synchronize our PC network.

To synchronize a mote with a PC in software, we adopted the underlying methodology of ETA and applied it to serial communication. On the mote, a timestamp is taken upon transfer of a synchronization byte and inserted into the outgoing message. On the PC, a timestamp is taken immediately after the UART issues the interrupt, and the PC regards the difference between these two timestamps as the PC-mote offset, ϕ_{mote}^{pc}. Serial communication bit rate between the mote and PC is 57600 baud, which approximately amounts to a transfer time of 139 microseconds per byte. However, the UART will not issue an interrupt to the CPU until its 16-byte buffer nears capacity or a timeout occurs. Because the synchronization message is six bytes, reception time in this case will consist of the transfer time of the entire message in addition to the timeout time and the time it takes to transfer the date from the UART buffer into main memory by the CPU. This time is compensated for by the receiver, and the clock offset between the two devices is determined as the difference between the PC receive time and the mote transmit time.

Architecture We have developed a PC-based time synchronization service for reactive data fusion applications in HSNs[4]. Figure 3 illustrates the interaction of each component within the service. The service collects sensor data from applications that run on the local PC, as well as from other service instances running on remote PCs. It accepts event messages on a specific port, converts the embedded timestamps to the local timescale, and forwards the messages toward the sensor-fusion node. To maintain synchronization with the rest of the PC network, the service uses RBS. The arrival times of the reference broadcasts are stored in a reference table and accessed for timescale transformation. In addition, the service accepts mote-based event messages, and converts the embedded timestamps using the ETA serial timestamp synchronization method

[4] Our synchronization service implementation is available as open source, and can be found at `http://www.isis.vanderbilt.edu/Projects/NEST/HSNTimeSync.html`.

outlined above. The messages are then forwarded toward the sensor-fusion node. The service instance that resides on the sensor-fusion node transforms incoming timestamps into its local timescale before passing the event messages up to the sensor-fusion application.

Kernel modifications in the serial and wireless drivers were required in order to take accurate timestamps. Upon receipt of a designated synchronization byte, the time is recorded and passed up to the synchronization service in the user-space. The mote implementation uses the TimeStamping interface, provided with the TinyOS distribution [19]. A modification was made to the UART interface to insert a transmission timestamp into the event message as it is being transmitted between the mote and PC. The timestamp is taken immediately before a synchronization byte is transmitted, then inserted at the end of the message.

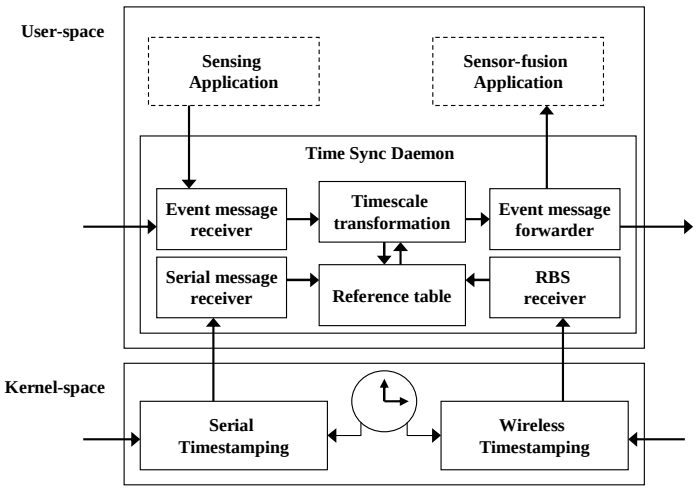

Fig. 3. PC-based time synchronization service.

7 Evaluation

Experimental Setup Our HSN testbed consists of seven Crossbow Mica2 motes and four stationary ActivMedia Pioneer robots with embedded Redhat Linux 2.4 PCs, as illustrated in Figure 4. In addition we employ a Linux PC to transmit RBS beacons. We chose this testbed because the issues that arise here are representative of practical HSN configurations such as hierarchical clustering and networks with multiple sinks. In addition, routing sensor data from a mote network to a PC base station is a dominant communication pathway in sensor network architectures.

The reference broadcast node transmits a reference beacon containing a sequence number once every ten seconds. The arrival of these messages are timestamped in the kernel and stored in a reference table. Simultaneously, a designated

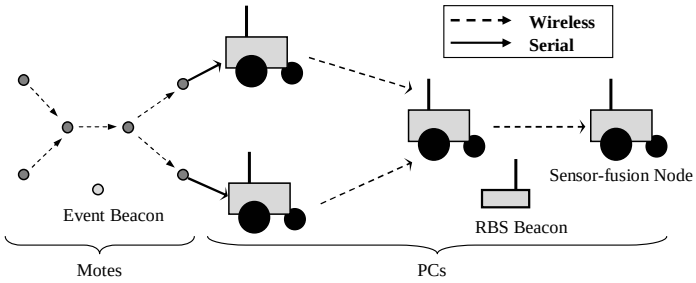

Fig. 4. Our sensor network testbed. Arrows indicate communication flow.

mote broadcasts event beacons, once every $4000 \pm \epsilon$ milliseconds, where ϵ is a random number in the range (0,1000). Six hundred event beacons are broadcast per experiment. The motes timestamp the arrival of the event beacon, and place the timestamp into a message. The message is routed over three hops in the mote network to the mote-PC gateways, using RITS to convert the timestamp to the local timescale with each hop. The message is next transferred from the mote network to the PC network over the mote-PC serial connections, and the event timestamp is converted to the timescale of the gateway PCs. The gateway PCs forward the message two additional hops to the sensor-fusion node. The experiment was repeated using the different clock skew compensation techniques in the PC network, as described in Section 5. Because RITS synchronizes a single sender with a single receiver at the time of data transfer, and because event data is forwarded to the base station immediately after the event is detected or a data message is received, clock skew compensation in the mote network provides negligible improvement.

Subsystem Synchronization Results We performed a series of experiments to quantify synchronization error on the individual critical paths in the HSN. The results allow us to justify our selection of synchronization protocols for the entire HSN. Note that no clock skew compensation was performed for these initial tests. To determine synchronization error, we used the *pairwise difference* evaluation method. Two nodes, N_1 and N_2, simultaneously timestamp the occurrence of an event, such as a reference beacon. These timestamps are then transformed to the timescale of node N_3, and the absolute value of their difference represents the error in the timescale transformation.

Experimental results in the literature (e.g. [8], [13]) indicate that RITS works well for synchronizing the mote network. We confirmed this on a 3-hop network of Mica2 motes. A beacon was broadcast to two outlying motes, which timestamped its arrival and forwarded the timestamp to a network sink node 3 hops away. At each intermediate node, the timestamps were converted to the local timescale. Synchronization error was calculated as timestamps arrived at the network sink node and, over 100 synchronizations, the average error was $7.27\mu s$, with a maximum of $37\mu s$.

Based on the implementation described in [5], we synchronized our PC network using RBS. We used a separate transmitter to broadcast a reference beacon every ten seconds (randomized) for 100 runs. Two PCs received reference broadcast r at local times $PC_1(t_r)$ and $PC_2(t_r)$, respectively. Synchronization error was $8.10\mu s$ on average, and $43\mu s$ maximum. Results are displayed in Figure 5a.

Figure 5b plots the synchronization error between two PCs using RITS. Every two seconds, PC_1 sent two synchronization messages to PC_2. Immediately before the command to send the first message was issued to the network interface controller on the sender, a transmission timestamp $PC_1(t_{tx})$ was taken in the kernel. This was found to be the latest possible time for the sender to take the timestamp. However, by the time the timestamp had been acquired, the message had already been transmitted, so a second message was needed to deliver the timestamp to the receiver. PC_2 recorded the timestamp $PC_2(t_{rx})$ in the kernel interrupt function upon receipt of the first message, and obtained the sender timestamp in the second message shortly after. The results show that we cannot expect consistent synchronization accuracy using RITS with the 802.11 networked Linux PCs. This is partly due to sender-side message delay error in RITS, which is nonexistent in RBS. In addition, the PC-based operating system is not tightly integrated with the network interface, and therefore the timestamping precision of the transmission and reception of sync bytes is degraded.

To synchronize the mote with the PC, we used the synchronization methodology described in Section 6. To evaluate synchronization accuracy, GPIO pins on the mote and PC were connected to an oscilloscope, and set high upon timestamping. The resulting output signals were captured and measured. The test was performed over 100 synchronizations, and resulting error was $7.32\mu s$ on average. The results are displayed in Figure 6. The majority of the error is due to nondeterministic message delay resulting from jitter, both in the UART and the CPU. A technique to compensate for such jitter on the motes is presented in [7], however, we did not attempt it on the PCs.

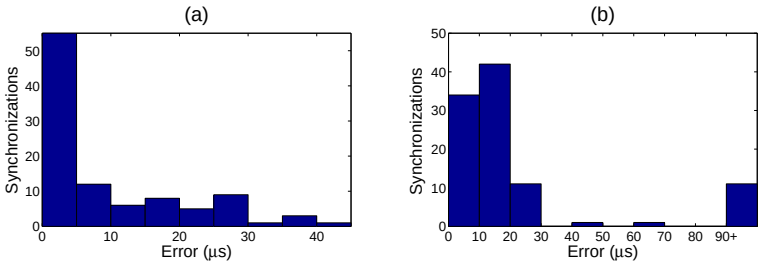

Fig. 5. (a) RBS and (b) RITS synchronization error between two PCs.

HSN Synchronization Results Figure 7 summarizes the synchronization error for each type of clock skew compensation technique under normal operating conditions, high network congestion, and high I/O load. To simulate a

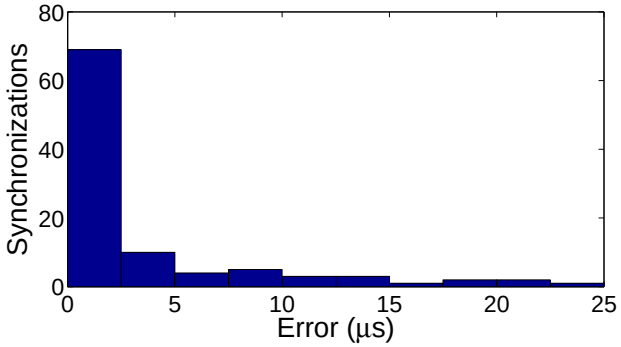

Fig. 6. Mote-PC error using ETA.

high network load, an extra mote and PC (not pictured in Figure 4) were introduced, each broadcasting messages of random size every 100 milliseconds. We first examined synchronization without clock skew compensation. The sensor-fusion node reported the average difference between the source timestamps as $12.05\mu s$, with a maximum of $270\mu s$. Next, we synchronized the PC network using linear regression as our clock skew compensation technique. Linear regression was implemented with a history size of 8 local-remote timestamp pairs for each neighbor, and the relative rate was initialized to 1. As expected, there was a notable improvement in synchronization accuracy, with an average of $7.62\mu s$ error, and a maximum of $84\mu s$. Repeating the experiment using exponential averaging gives errors similar to linear regression. For exponential averaging, we chose a value of 0.10 for α, and initialized the average relative rate to 1. These values were determined experimentally for rapid synchronization convergence. The average error recorded was $8.82\mu s$, with a maximum of $112\mu s$. The average synchronization error using phase-locked loops was $7.85\mu s$, with a maximum of $196\mu s$. For the digital loop filter, we used gains of $K_1 = 0.1$ and $K_2 = 0.09$, determined experimentally.

Memory overhead is minimal for each clock skew compensation technique. Nodes require 8 bytes for the current relative rate estimate for each neighbor within single-hop range. In the case of linear regression, a small history buffer for each neighbor is also required. Our implementation has no message overhead (except for the beacons transmitted by the RBS server). In fact, the only modification to the data message is the addition of a four-byte timestamp. Because these synchronization timestamps piggyback on data messages, no additional messages are required. Our methodology is therefore energy efficient, because message overhead directly impacts energy consumption. Convergence time depends on input parameters to the clock skew compensation algorithm. We found that, on average, it took the network 80 seconds to synchronize with linear regression, 200 seconds with exponential averaging, and 1300 seconds with phase-

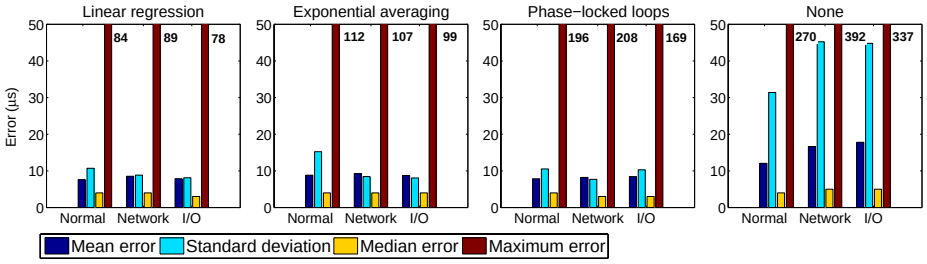

Fig. 7. HSN synchronization error with different types of clock skew compensation under normal operating conditions (Normal), high network congestion (Network), and high I/O load (I/O).

locked loops. Note that these convergence times reflect an inter-beacon delay of four seconds.

Discussion These results show that we are able to achieve microsecond-accurate synchronization in the HSN. Because error accrues with each timescale transformation, achieving this level of synchronization accuracy over a 6-hop network of heterogeneous devices is significant. Although the maximum synchronization error extends to tens of microseconds, it was principally caused by a small number of synchronization attempts in which prolonged operating system interrupt operations occurred. Although these were difficult to avoid, they did not occur frequently, and the worst-case synchronization error for each type of clock skew compensation technique was acceptable for most kinds of HSN data fusion applications. Furthermore, time synchronization was not affected by the most common types of nondeterministic behavior, such as network congestion and I/O operations.

The accuracy of the mote-PC synchronization is quite good. Although we did see error in the tens of microseconds, the maximum did not exceed $50\mu s$, and the average was below $10\mu s$. This is significant, because it demonstrates that microsecond accuracy synchronization can be achieved between mote and PC networks, enabling the development of HSN applications that require precision time synchronization. In addition, because this technique uses UART communication, it can easily be adapted for synchronization with UART-supported peripheral sensing devices.

8 Conclusion

Time synchronization is an important and necessary component in most wireless sensor network applications. However, as we describe in this paper, its implementation is non-trivial, especially when high-precision synchronization is required. In networks of heterogeneous devices, the problem is compounded by issues of system integration, and therefore alternative techniques must be employed to reduce synchronization error. We have shown that with certain configurations, microsecond accuracy can be achieved with careful selection of hardware, software, and network components. Our methodology is generally portable to other

platforms, provided mechanisms exist within the target configuration that enable low-level timestamping.

Acknowledgements This work was supported in part by a Vanderbilt University Discovery Grant, ARO MURI grant W911NF-06-1-0076, NSF CAREER award CNS-0347440, and NSF grant CNS-0721604. The authors would also like to thank Manish Kushwaha and Janos Sallai for their help with this project.

References

1. Williams, S.M., Frampton, K.D., Amundson, I., Schmidt, P.L.: Decentralized acoustic source localization in a distributed sensor network. Applied Acoustics **67** (2006)
2. Ledeczi, A., Nadas, A., Volgyesi, P., Balogh, G., Kusy, B., Sallai, J., Pap, G., Dora, S., Molnar, K., Maroti, M., Simon, G.: Countersniper system for urban warfare. ACM Transactions on Sensor Networks **1**(2) (2005)
3. Huang, M., Jiang, L., Liaw, P.K., Brooks, C.R., Seeley, R., Klarstrom, D.L.: Using acoustic emission in fatigue and fracture materials research. JOM **50**(11) (1998)
4. Romer, K.: Time synchronization in ad hoc networks. In: ACM MobiHoc. (2001)
5. Elson, J., Girod, L., Estrin, D.: Fine-grained network time synchronization using reference broadcasts. In: OSDI. (2002)
6. Ganeriwal, S., Kumar, R., Srivastava, M.B.: Timing-sync protocol for sensor networks. In: ACM SenSys. (2003)
7. Maroti, M., Kusy, B., Simon, G., Ledeczi, A.: The flooding time synchronization protocol. In: ACM SenSys. (2004)
8. Sallai, J., Kusy, B., Ledeczi, A., Dutta, P.: On the scalability of routing integrated time synchronization. In: EWSN. (2006)
9. Yarvis, M., Kushalnagar, N., Singh, H., Rangarajan, A., Liu, Y., Singh, S.: Exploiting heterogeneity in sensor networks. In: IEEE Infocom. (2005)
10. Duarte-Melo, E., Liu, M.: Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks. In: IEEE Globecom. (2002)
11. Lazos, L., Poovendran, R., Ritcey, J.A.: Probabilistic detection of mobile targets in heterogeneous sensor networks. In: IPSN. (2007)
12. Romer, K., Blum, P., Meier, L.: Time synchronization and calibration in wireless sensor networks. In Stojmenovic, I., ed.: Wireless Sensor Networks. Wiley and Sons (2005)
13. Kusy, B., Dutta, P., Levis, P., Maroti, M., Ledeczi, A., Culler, D.: Elapsed time on arrival: A simple and versatile primitive for time synchronization services. International Journal of Ad hoc and Ubiquitous Computing **2**(1) (2006)
14. Girod, L., Bychkovsky, V., Elson, J., Estrin, D.: Locating tiny sensors in time and space: A case study. In: ICCD: VLSI in Computers and Processors. (2002)
15. Polastre, J., Hill, J., Culler, D.: Versatile low power media access for wireless sensor networks. In: ACM SenSys. (2004)
16. Elson, J., Romer, K.: Wireless sensor networks: A new regime for time synchronization. In: HotNets-I. (2002)
17. Mills, D.L.: Modelling and analysis of computer network clocks. Technical Report 92-5-2, Electrical Engineering Department, University of Delaware (1992)
18. D. L. Mills: Internet time synchronization: The network time protocol. IEEE Transactions on Communications **39**(10) (1991)
19. Levis, P., Madden, S., Gay, D., Polastre, J., Szewczyk, R., Woo, A., Brewer, E., Culler, D.: The emergence of networking abstractions and techniques in TinyOS. In: NSDI. (2004)