Title

Short Paper: A Wireless Time-Synchronized COTS Sensor Platform, Part I: System Architecture

Authors

Elson, J
Girod, Lewis
Estrin, D

Publication Date

2002-09-01

Peer reviewed

SHORT PAPER: A WIRELESS TIME-SYNCHRONIZED COTS SENSOR PLATFORM PART I: SYSTEM ARCHITECTURE

Jeremy Elson, Lewis Girod, and Deborah Estrin

Department of Computer Science, 3440 Boelter Hall
University of California, Los Angeles USA 90095
{jelson,girod,destrin}@lecs.cs.ucla.edu

ABSTRACT

In this paper, we describe an implementation of a ad-hoc, distributed sensor platform that provides synchronized time to its users. By abstracting the time synchronization layer away, we allow developers to focus on the core challenges of their applications (e.g., signal processing, aggregation, routing) rather than dealing with the algorithmic and systems issues that inevitably arise when integrating sensing with distributed synchronization. Through a variety of techniques, notably the use of Reference-Broadcast Synchronization (RBS) [5], our platform offers better than 5μsec precision when comparing streams of audio data sampled at nodes separated by one network hop.

1. INTRODUCTION

Recent advances in miniaturization and low-cost, low-power design have led to active research in large-scale, highly distributed systems of small, wireless, low-power, unattended sensors and actuators [2]. While individual sensor nodes have only limited functionality, the global behavior of a sensor network can be quite complex. The network's value is in this *emergent behavior*: the functionality of the whole is greater than the sum of its parts. Such behavior is achieved, in part, through *data fusion*, the process of merging individual sensor readings into a high-level result.

Time synchronization is critical for distributed data fusion; it is nearly always required to meaningfully correlate the output of distributed sensors. There are many examples of sensor network tasks that require synchronized time: for example, to save energy by forming a sleep and wakeup schedule [3]; create a distributed acoustic beamforming platform [15]; integrate a time-series of proximity detections into a velocity estimate [4]; measure the time-of-flight of sound for localizing its source [7]; or suppress redundant messages by recognizing that they describe duplicate detections of the same event by different by nearby sensors [9].

In this paper, we describe an implementation of a distributed sensor platform that provides synchronized time to its users. By abstracting the time synchronization layer away, we allow developers to focus on the core challenges of their applications (e.g., signal processing, aggregation, routing) rather than dealing with the algorithmic and systems issues that inevitably arise when integrating sensing with distributed synchronization. Through a variety of techniques, most notably the use of Reference-Broadcast Synchronization (RBS) [5], our platform offers better than 5μsec precision when comparing streams of audio data acquired at peers separated by one hop.

Of course, time synchronization is a well-studied problem; for decades, protocols such as NTP [12] have kept the Internet's clocks ticking in phase. However, in a network where physical-layer broadcasts are possible, RBS can improve precision over NTP by nearly an order of magnitude [5]. In addition, as we argue in [6], NTP is not necessarily the right choice for a distributed sensor networks for more subtle reasons.

The organization of this paper is as follows. In Section 2, we describe our hardware platform, and the advantages of using a commercial off-the-shelf (COTS) system—a Compaq iPAQ—rather than something custom-made. Section 3 how the integrated 802.11 (wireless Ethernet) network can be used to achieve inter-node time synchronization of better than 5μsec. In Section 4, we discuss our audio server, which simplifies application development and integrates acoustic sampling with the time synchronization system. Finally, Section 5 has our conclusions and future directions. Part II of this paper [15] describes an application implemented on our platform: a distributed, acoustic beamforming array, based on earlier work in centralized beamforming [16].

2. HARDWARE PLATFORM

Over the past few years, a number of hardware platforms have emerged in the wireless network sensor arena, perhaps

In *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, California. September 2002.

starting with the WINS system developed at UCLA and Rockwell [3, 14]. Most recently, the Berkeley Mote platform [8, 10] is notable due to its growing popularity in the research community. While these platforms are typically the most energy-efficient and have the smallest form factors, we have also found COTS (commercial off-the-shelf) platforms to be useful due to their lower cost, ready availability in quantity, and ease of use (minimal "hardware hacking" required). While the dedicated platforms are usually required for real world deployments, a COTS platform facilitates rapid application prototypes and data collection.

Our COTS sensor platform is based around the Compaq iPAQ 3760, which is a handheld, battery-powered device normally meant to be used as a personal organizer (PDA). The iPAQ provides a reasonable balance of cost, availability, and functionality. It has a 206MHz Intel StrongARM-1110 processor, 64MB of RAM, and 32MB of persistent FLASH. Our iPAQs use the "Familiar" distribution of Linux [1], which provides a development environment with conveniences typically not found in embedded operating systems (e.g., remote login for debugging, network error logging, network software upgrade, etc.). We also selected the iPAQ because it is readily available COTS hardware, has reasonable battery life, and is usable right off the shelf. The standard model has a built-in speaker and microphone, with an audio codec capable of sampling at 48KHz. Finally, the iPAQs have a serial port, and a PCMCIA bus, for which a wide variety of peripherals are available. All of our iPAQs have spread-spectrum wireless Ethernet cards (802.11b direct sequence, 11Mbit/sec), which can operate in either base-station or ad-hoc mode.

3. NETWORK TIME SYNCHRONIZATION

In our system, network time synchronization is accomplished using an implementation of Reference-Broadcast Synchronization, or RBS, described in more detail in [5]. Briefly, the fundamental property of RBS is that it uses the physical-layer broadcast property of wireless networks to *synchronize a set of receivers with one another*. By using only receiver-to-receiver relations, the largest sources of nondeterministic latency are removed from the critical path. This results in significantly better-precision synchronization than traditional algorithms that synchronize a sender with a receiver with a correction for the measured round-trip delay. In addition, because the residual error is often a well-behaved distribution (e.g., Gaussian), multiple reference broadcasts can be sent over time, allowing both improved precision of the phase offset estimate and correction for clock skew.

Complex disciplines exist that can lock an oscillator's phase and frequency to an external standard [13]. However, we selected a very simple yet effective algorithm to correct skew: a *least-squares linear regression* on the time series

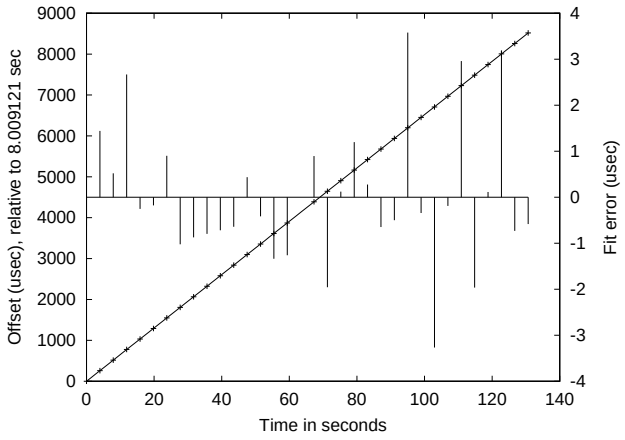

Figure 1. Example of relative clock phase and frequency recovery using a time series of compared broadcast observation times. using a linear regression through packet observations. The diagonal line is the best linear fit to the data—i.e., the line that minimizes RMS error. The vertical impulses, read with respect to the right-hand y axis, show the distance from each point to the best-fit line.

of phase differences between nodes, after automatic outlier rejection. This offers a fast, closed-form method for finding the best fit line through the phase observations over time. The frequency and phase of the local node's clock with respect to the remote node can be recovered from the slope and intercept of the line. A visualization of this procedure is shown in Figure 1.

Our RBS daemon simultaneously acts in both "sender" and "receiver" roles. Every 10 seconds (slightly randomized to avoid unintended synchronization), each daemon emits a pulse packet with a sequence number and sender ID. The daemon also watches for such packets to arrive; it timestamps them and periodically sends a report of these timestamps back to the pulse sender along with its receiver ID. The pulse sender collects all of the pulse reception reports and computes clock conversion parameters between each pair of nodes that heard its broadcasts. These parameters are then broadcast back to local neighbors. The RBS daemons that receive these parameters make them available to users. (RBS never sets the nodes' clocks, but rather provides a user library that converts UNIX `timeval`s from one node ID to another.)

To improve the accuracy of the packet timestamps, we made use of the Berkeley packet capture library, `libpcap` [11]. Without `libpcap`, typical applications read the system clock from user-space, after the process has been notified an event (such as a packet arrival). This can reduce the precision of timestamps due to the scheduling and task-switching latency between the event and the time of

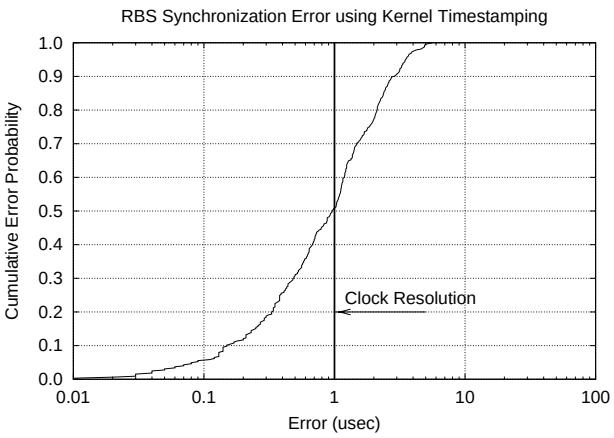

Figure 2. Cumulative distribution function of RBS synchronization error on our Compaq iPAQs. Mean error was 1.26μsec, standard deviation of 1.11μsec. 50% of trials were within 0.96μsec error; 95% within 3.51μsec, and 99% within 4.89μsec.

the clock read. In contrast, `libpcap` can provide precise packet-arrival timestamps by coöperating with the Linux kernel, which annotates packets with the reception times in the network interface's interrupt handler. The timestamp (and other meta-data) is then passed up to the application along with the packet itself. This technique allows the system to maintain high precision timestamps, even at times of high CPU contention.

To test the precision of the time synchronization on our platform, we connected a GPIO output from each of two IPAQs to an external logic analyzer. The analyzer was programmed to report the time difference between two pulses seen on each of its input channels. In each trial, we used RBS' clock conversion parameters to command each IPAQ to raise their GPIO lines high at the same time. We ran a total of 325 trials, each separated by about 8 seconds, for a total test period of about 45 minutes. The results are shown in Figure 2; RBS achieved a mean $1.26 \pm 1.11\mu$sec synchronization error. We believe this result was primarily limited by the iPAQ's clock resolution under Linux, which is 1μsec.

4. AUDIO SAMPLING SERVER

The second system service that facilitates distributed acoustic sampling on our platform is the *audio server*. This is a process that continuously reads samples from the audio codec, buffering the most recent 10 seconds, and making that buffer available to user applications on demand. This scheme can dramatically decrease the complexity of user applications in a number of ways. For example, it enables systems with external triggers (e.g., seismic sensors or cameras), which can easily go back in time and retrieve the audio data that corresponded to the trigger event. We have also found this "after-the-fact sampling" useful for reducing the complexity of protocol interactions—for example, in our acoustic ranging application [7]. Early designs required several round-trips worth of interaction to ensure the receiver would be recording in advance of the sender's chirp. By recording continuously, the protocol is simplified: "I chirped 2 seconds ago, when did you receive it?" The audio server is also an effective way to resolve contention for the codec in applications where multiple processes all need access to the incoming audio stream.

The audio server serves another function: it improves the timing precision of the recorded audio data. Unfortunately, cheap consumer-grade audio codecs such as those found in a PDA tend to have nondeterministic latency before the beginning of a recording or playback. It is important to minimize these effects since they contribute directly to synchronization error between the audio streams. We have found that this problem can be avoided by running the codec continuously. With the help of a modification to the Linux kernel's codec device driver, the audio server timestamps each DMA transfer of audio samples from the codec's chipset as it arrives. Although there is a delay between when audio is physically acquired and when the DMA transfer completes, this delay appears to be deterministic and easily calibrated out. In contrast, the nondeterministic delay between a "start recording now" command and the first audio sample leads to poor synchronization.

Our time synchronization daemon, described in its RBS role in Section 3, also supports synchronization between components *within* a system. Components such as the audio codec have sampling clocks that are independent of the system clock. Network RBS only synchronizes the *system* clocks, but applications typically need the sampling clocks to be synchronized so that audio data can be correlated across nodes. We therefore explicitly model *both* types of clocks in our synchronization process, with explicit conversion parameters between them. Each time the audio server receives and timestamps a new block of audio data, it injects a pair of values into the time daemon: an audio sample number and associated system clock value. These pairs relate the two clocks to each other over time; the daemon computes conversion parameters that allow iPAQ system clock values to be converted to audio sample numbers, and vice-versa. The daemon uses the same linear least-squares regression and outlier rejection as it does for RBS. This makes it very robust against outliers due to (for example) an occasional late DMA transfer.

By integrating this "codec pairs synchronization" with the RBS synchronization described in Section 3, exact audio samples can be correlated across iPAQs. That is, a codec sample number on iPAQ X is converted to an X system time

value; X's system time is converted to a system time on Y using RBS; and the codec pairs are used again to convert the Y system clock value to a Y codec sample number. The codecs sample at 48KHz, or $\approx 21\mu$sec per sample, so the 5μsec inter-node synchronization achieved by RBS is well within the threshold needed to correlate co-occurring audio samples from codecs on a distributed set of iPAQs.

5. CONCLUSIONS

In this paper, we have presented a PDA-based COTS platform that provides time distributed synchronization for both the system clocks and the audio codecs, enabling a variety of interesting distributed sensing experiments. (Part II of this paper [15] describes an application implemented on our platform: a distributed, acoustic beam-forming array, based on earlier work in centralized beam-forming [16].) Our platform typically achieves better than 5μsec phase error and samples audio at 48KHz.

While dedicated platforms are likely to be preferred for real-world sensor deployment, our COTS platform is attractive for early prototyping and data collection. It has a general purpose operating system (Linux), allowing a wide range of application exploration. The platform is sufficiently small, low-power, general-purpose, and time-synchronized to gain valuable experience through pilot deployments and real-world data collection experiments.

Our system image is publicly available. For more information, please see `http://lecs.cs.ucla.edu`, or contact the authors (`testbed@lecs.cs.ucla.edu`).

Acknowledgements

This work was made possible with support from the DARPA NEST program (the "GALORE" project, grant F33615-01-C-1906). Additional support was provided through the University of California MICRO program (grant number 01-031) and matching funds from Intel Corporation.

6. REFERENCES

[1] Familiar Linux. http://www.handhelds.org.

[2] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38(4):393–422, March 2002.

[3] G. Asada, T. Dong, F. Lin, G. Pottie, W. Kaiser, and H. Marcy. Wireless integrated network sensors: Low power systems on a chip. In *Proceedings of European Solid State Circuits Conference, The Hague, Netherlands*, October 1998.

[4] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. Habitat monitoring: Application driver for wireless communications technology. In *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001. Available at http://www.isi.edu/scadds/papers/CostaRica-oct01-final.ps.

[5] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. Technical Report UCLA-CS-020008, University of California Los Angeles, May 2002. http://lecs.cs.ucla.edu/Publications.

[6] Jeremy Elson and Kay Römer. Wireless sensor networks: A new regime for time synchronization. Technical report, UCLA Technical Report, July 2002. http://lecs.cs.ucla.edu/Publications.

[7] Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah Estrin. Locating tiny sensors in time and space: a case study. In *Proceedings of the International Conference on Computer Design (ICCD)*, Freiburg, Germany, September 2002. http://lecs.cs.ucla.edu/Publications.

[8] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth International Conference on Arhitectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.

[9] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking*, pages 56–67, Boston, MA, August 2000. ACM Press.

[10] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Next century challenges: mobile networking for Smart Dust. In *Proceedings of the fifth annual ACM/IEEE Intl. Conf. on Mobile computing and networking*, pages 271–278, 1999.

[11] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In USENIX Association, editor, *Proceedings of the Winter 1993 USENIX Conference: January 25–29, 1993, San Diego, California, USA*, pages 259–269, Berkeley, CA, USA, Winter 1993. USENIX.

[12] David L. Mills. Internet Time Synchronization: The Network Time Protocol. In Zhonghua Yang and T. Anthony Marsland, editors, *Global States and Time in Distributed Systems*. IEEE Computer Society Press, 1994.

[13] David L. Mills. Adaptive hybrid clock discipline algorithm for the network time protocol. *IEEE/ACM Transactions on Networking*, 6(5):505–514, October 1998.

[14] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, May 2000.

[15] H. Wang, L. Yip, D. Maniezzo, J.C. Chen, R.E. Hudson, J.Elson, and K.Yao. A Wireless Time-Synchronized COTS Sensor Platform Part II–Applications to Beamforming. In *Proceedings of IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, CA, September 2002.

[16] K. Yao, R.E. Hudson, C.W. Reed, D. Chen, and F. Lorenzelli. Blind beamforming on a randomly distributed sensor array system. *IEEE Journal of Selected Areas in Communications*, 16(8):1555–1567, Oct 1998.