

USENIX Association

Proceedings of the
5th Symposium on Operating Systems
Design and Implementation

Boston, Massachusetts, USA
December 9–11, 2002



© 2002 by The USENIX Association

All Rights Reserved

For more information about the USENIX Association:

Phone: 1 510 528 8649

FAX: 1 510 548 5738

Email: office@usenix.org

WWW: <http://www.usenix.org>

Rights to individual papers remain with the author or the author's employer.

Permission is granted for noncommercial reproduction of the work for educational or research purposes.

This copyright notice must be included in the reproduced paper. USENIX acknowledges all trademarks herein.

Fine-Grained Network Time Synchronization using Reference Broadcasts

Jeremy Elson, Lewis Girod and Deborah Estrin

Department of Computer Science, University of California, Los Angeles
{jelson,girod,destrin}@cs.ucla.edu

Abstract

Recent advances in miniaturization and low-cost, low-power design have led to active research in large-scale networks of small, wireless, low-power sensors and actuators. Time synchronization is critical in sensor networks for diverse purposes including sensor data fusion, coordinated actuation, and power-efficient duty cycling. Though the clock accuracy and precision requirements are often stricter than in traditional distributed systems, strict energy constraints limit the resources available to meet these goals.

We present *Reference-Broadcast Synchronization*, a scheme in which nodes send reference beacons to their neighbors using physical-layer broadcasts. A reference broadcast does not contain an explicit timestamp; instead, receivers use its arrival time as a point of reference for comparing their clocks. In this paper, we use measurements from two wireless implementations to show that removing the sender’s nondeterminism from the critical path in this way produces high-precision clock agreement ($1.85 \pm 1.28\mu\text{sec}$, using off-the-shelf 802.11 wireless Ethernet), while using minimal energy. We also describe a novel algorithm that uses this same broadcast property to federate clocks across broadcast domains with a slow decay in precision ($3.68 \pm 2.57\mu\text{sec}$ after 4 hops). RBS can be used without external references, forming a precise relative timescale, or can maintain microsecond-level synchronization to an external timescale such as UTC. We show a significant improvement over the Network Time Protocol (NTP) under similar conditions.

1 Introduction

Time synchronization is a critical piece of infrastructure for any distributed system. For years, NTP (the Net-

work Time Protocol) [24] has kept the Internet’s clocks ticking in phase. However, a new class of networks is emerging. Recent advances in miniaturization and low-cost, low-power design have led to active research in large-scale networks of small, wireless, low-power sensors and actuators [1]. These networks require that time be synchronized more precisely than in traditional Internet applications—sometimes on the order of $1\mu\text{sec}$ —due to their close coupling with the physical world and their energy constraints. For example, precise time is needed to measure the time-of-flight of sound [9, 22]; distribute a beamforming array [34]; form a low-power TDMA radio schedule [2]; integrate a time-series of proximity detections into a velocity estimate [4]; or suppress redundant messages by recognizing duplicate detections of the same event by different sensors [13]. In addition to these domain-specific requirements, sensor network applications often rely on synchronization as typical distributed systems do: for secure cryptographic schemes, coordination of future action, ordering logged events during system debugging, and so forth.

Many network synchronization algorithms have been proposed over the years, but most share the same basic design: a server periodically sends a message containing its current clock value to a client. Our work explores a form of time synchronization that differs from the traditional model. The fundamental property of our design is that it *synchronizes a set of receivers with one another*, as opposed to traditional protocols in which senders synchronize with receivers. In our scheme, nodes periodically send beacon messages to their neighbors using the network’s physical-layer broadcast. Recipients use the message’s arrival time as a point of reference for comparing their clocks. The message contains no explicit timestamp, nor is it important exactly when it is sent. We call this scheme *Reference-Broadcast Synchronization*, or RBS.

In this paper, we use measurements from two wireless implementations to show that removing the sender’s

nondeterminism from the critical path in this way results in a dramatic improvement in synchronization over using NTP. We also present an algorithm that allows time to be propagated across broadcast domains without losing the reference-broadcast property. In this way, nodes in a multi-hop network can form a highly precise relative timescale, or maintain microsecond-level synchronization to an external timescale such as UTC.

The most significant limitation of RBS is that it requires a network with a physical broadcast channel. It can not be used, for example, in networks that employ point-to-point links. However, it is applicable for a wide range of applications in both wired and wireless networks where a broadcast domain exists and higher-precision or lower-energy synchronization is required than what NTP can typically provide.

The organization of this paper is as follows: We first review related work in Section 2. In Section 3, we describe the design of traditional time synchronization protocols in more detail, and contrast it to the RBS design philosophy. In Section 4, we describe the basic building blocks of RBS, including estimation of phase offset (§4.1) and clock skew (§4.2). We also present empirical data from two wireless implementations of single-hop RBS (§4.3–§4.5). In Section 5, we present a novel algorithm for extending RBS across broadcast domains with slow precision decay. We also describe empirical results from a multi-hop RBS implementation (§5.3). In Section 6, we describe applications from three research groups that use RBS. Finally, we offer our conclusions and describe our plans for future work in Section 7.

2 Related Work

A landmark paper in computer clock synchronization is Lamport’s work that elucidates the importance of *virtual clocks* in systems where causality is more important than absolute time [18]. Though Lamport’s work focused on giving events a total order rather than quantifying the time difference between them, it has emerged as an important influence in sensor networks. Many sensor applications require only relative time, for example timing the propagation delay of sound [9], and thus absolute time may not be needed.

Over the years, many protocols have been designed for maintaining synchronization of physical clocks over computer networks [5, 10, 24, 30]. These protocols all have basic features in common: a simple connection-

less messaging protocol; exchange of clock information among clients and one or more servers; methods for mitigating the effects of nondeterminism in message delivery and processing; and an algorithm on the client for updating local clocks based on information received from a server. They do differ in certain details: whether the network is kept internally consistent or synchronized to an external standard; whether the server is considered to be the canonical clock, or merely an arbiter of client clocks, and so on.

Mills’ NTP [24] stands out by virtue of its scalability, self-configuration in large multi-hop networks, robustness to failures and sabotage, and ubiquitous deployment. NTP allows construction of a hierarchy of time servers, multiply rooted at canonical sources of external time.

Synchronization to an external timescale is typically provided by the Global Positioning System (GPS), a constellation of satellites operated by the U.S. Department of Defense [16]. Commercial GPS receivers can achieve accuracy of better than 200nsec relative to UTC [20]. However, GPS requires a clear sky view, which is unavailable in many application areas—for example, inside of buildings, beneath dense foliage, underwater, on Mars, or behind enemy lines where GPS is jammed. In addition, receivers can require several minutes of settling time, and may be too large, costly, or high-power to justify on a small, cheap sensor node.

Perhaps most closely related to our work are the CesiumSpray system [32] (the foundations of which were described by Veríssimo and Rodrigues [31]), and the 802.11-based broadcast synchronization described by Mock *et al.* [27]. Their systems, like ours, make use of the inherent properties of broadcast media to achieve superior precision. However, their methods require all nodes to lie within a single broadcast domain; multiple domains can not be federated, except by depending on an out of band common view of time (e.g., GPS). In contrast, RBS incorporates a novel multi-hop algorithm; in Section 5, we show it can maintain tight time synchronization across many hops through a wireless network without external infrastructure support. In addition, we have fully decoupled the sender from the receiver—only synchronizing receivers with one another, even when relating the timescale to an external timescale such as UTC. CesiumSpray retains a dependence on the relationship between sender and receiver when synchronizing nodes to UTC.

Liao *et al.* describe a system for synchronizing interfaces on a system-area network, Myrinet, with microsecond

accuracy [19]. Although the precision bound in this system is similar to ours, their results depend on the use of an underlying network that offers latency and determinism guarantees in fixed topologies. In contrast, our scheme works over a much broader class of networks, over a much wider area, and does not require a tight coupling between the sender and its network interface.

Similarly, on Berkeley Mote hardware, Hill *et al.* report $2\mu\text{sec}$ precision for receivers within a single broadcast domain [11]; Ganerival *et al.* report $25\mu\text{sec}$ -per-hop precision across multiple hops [7]. Both of these results are achieved by tightly integrating the MAC with the application, and building a deterministic bit-detector into the MAC layer. RBS does not require that the application be collapsed into the MAC, and indeed will be shown in Section 4.4 to work on commodity hardware (802.11). However, their work is complementary to ours, as our focus is not on building deterministic detectors, but rather making the best use of existing detectors. By leveraging Hill's $0.25\mu\text{sec}$ -precision bit detector, RBS could achieve several times the precision of their scheme, based on their $2\mu\text{sec}$ jitter budget analysis.

3 Traditional Synchronization Methods

Before discussing RBS in detail, we describe how existing time synchronization protocols typically work, and their usual sources of error.

Many synchronization protocols exist, and most share a basic design: a server periodically sends a message containing its current clock value to a client. A simple one-way message suffices if the typical latency from server to client is small compared to the desired accuracy. A common extension is to use a client request followed by a server's response. By measuring the total round-trip-time of the two packets, the client can estimate the one-way latency. This allows for more accurate synchronization by accounting for the time that elapses between the server's creation of a timestamp and the client's reception of it. NTP is a ubiquitously adopted protocol for Internet time synchronization that exemplifies this design.

3.1 Sources of Time Synchronization Error

The enemy of precise network time synchronization is *non-determinism*. Latency estimates are confounded by random events that lead to asymmetric round-trip mes-

sage delivery delays; this contributes directly to synchronization error. To better understand the source of these errors, it is useful to decompose the source of a message's latency. Kopetz and Schwabl characterize it as having four distinct components [17]:

- *Send Time*—the time spent at the sender to construct the message. This includes kernel protocol processing and variable delays introduced by the operating system, e.g. context switches and system call overhead incurred by the synchronization application. Send time also accounts for the time required to transfer the message from the host to its network interface.
- *Access Time*—delay incurred waiting for access to the transmit channel. This is specific to the MAC protocol in use. Contention-based MACs (e.g., Ethernet [23]) must wait for the channel to be clear before transmitting, and retransmit in the case of a collision. Wireless RTS/CTS schemes such as those in 802.11 networks [14] require an exchange of control packets before data can be transmitted. TDMA channels [29] require the sender to wait for its slot before transmitting.
- *Propagation Time*—the time needed for the message to transit from sender to receivers once it has left the sender. When the sender and receiver share access to the same physical media (e.g., neighbors in an ad-hoc wireless network, or on a LAN), this time is very small as it is simply the physical propagation time of the message through the media. In contrast, Propagation Time dominates the delay in wide-area networks, where it includes the queuing and switching delay at each router as the message transits through the network.
- *Receive Time*—processing required for the receiver's network interface to receive the message from the channel and notify the host of its arrival. This is typically the time required for the network interface to generate a message reception signal. If the arrival time is timestamped at a low enough level in the host's operating system kernel (e.g., inside of the network driver's interrupt handler), the Receive Time does not include the overhead of system calls, context switches, or even the transfer of the message from the network interface to the host.

Existing time synchronization algorithms vary primarily in their methods for estimating and correcting for these sources of error. For example, Cristian observed that

performing larger numbers of request/response experiments will make it more likely that at least one trial will not encounter random delays [5]. This trial, if it occurs, is easily identifiable as the shortest round-trip time. NTP filters its data using a variant of this heuristic.

Our scheme takes a different approach to reducing error. Instead of estimating error, we exploit the broadcast channel available in many physical-layer networks to remove as much of it as possible from the critical path. Our contributions in this paper stem from the observation that a message that is broadcast at the physical layer will arrive at a set of receivers with very little variability in its delay. Although the Send Time and Access Time may be unknown, and highly variable from message to message, the nature of a broadcast dictates that for a *particular* message, these quantities are *the same for all receivers*. This observation was also made by Veríssimo and Rodrigues [31], and later became central to their CesiumSpray system [32].

The fundamental property of Reference-Broadcast Synchronization is that a broadcast message is only used to *synchronize a set of receivers with one another*, in contrast with traditional protocols that synchronize the sender of a message with its receiver. Doing so removes the Send Time and Access Time from the critical path, as shown in Figure 1. This is a significant advantage for synchronization on a LAN, where the Send Time and Access time are typically the biggest contributors to the nondeterminism in the latency. Mills attributes most of the phase error seen when synchronizing an NTP client workstation to a GPS receiver on the same LAN (500 μ sec–2000 μ sec in his 1994 study [25]) to these factors—Ethernet jitter and collisions.

To counteract these effects, an RBS broadcast is always used as a *relative* time reference, never to communicate an absolute time value. It is exactly this property that eliminates error introduced by the Send Time and Access Time: each receiver is synchronizing to a reference packet which was, by definition, injected into the physical channel at the same instant. The message itself does not contain a timestamp generated by the sender, nor is it important exactly when it is sent. In fact, the broadcast does not even need to be a dedicated timesync packet. Almost any extant broadcast can be used to recover timing information—for example, ARP packets in Ethernet, or the broadcasted control traffic in wireless networks (e.g., RTS/CTS exchanges or route discovery packets).

3.2 Characterizing the Receiver Error

Previously, we described the four sources of latency in sending a message. Two of these—the Send Time and Access Time—are dependent on factors such as the instantaneous load on the sender’s CPU and the network. This makes them the most nondeterministic and difficult to estimate. As described above, RBS eliminates the effect of these error sources altogether; the two remaining factors are the Propagation Time and Receive Time.

For our purposes, we consider the Propagation Time to be effectively 0. The propagation speed of electromagnetic signals through air is close to c ,¹ and through copper wire about $\frac{2}{3}c$. For a LAN or ad-hoc network spanning tens of feet, propagation time is at most tens of nanoseconds, which does not contribute significantly to our μ sec-scale error budget. (In sensor networks, the error budget is often driven by the need to sense phenomena, such as sound, that move much more slowly than the RF-pulse used to synchronize the observers.) In addition, RBS is only sensitive to the *difference* in propagation time between a pair of receivers, as depicted in Figure 1.

The length of a bit gives us an idea of the Receive Time. For a receiver to interpret a message at all, it must be synchronized to the incoming message within one bit time. Latency due to processing in the receiver electronics is irrelevant as long as it is *deterministic*, since our scheme is only sensitive to *differences* in the receive time of messages within a set of receivers.² In addition, the system clock can easily be read at interrupt time when a packet is received; this removes delays due to receiver protocol processing, context switches, and interface-to-host transfer from the critical path.

To verify our assumptions about expected receiver error, we turned to our laboratory’s wireless sensor network testbed [4]. Specifically, we tested the Berkeley Mote, a postage-stamp sized narrowband radio and sensor platform developed by Pister *et al.* at Berkeley [15]. Motes use a minimal event-based operating system developed by Hill *et al.* specifically for that hardware platform called TinyOS [12]. We programmed 5 Motes to raise a GPIO pin high upon each packet arrival, and attached those signal outputs to an external logic analyzer that recorded the time of the packet reception events. An additional Mote emitted 160 broadcast packets over

¹A convenient rule of thumb is 1nsec/foot.

²It is important to consider effects of any intentional nondeterminism on the part of a receiver, such as aggregation of packets in a queue before raising an interrupt.

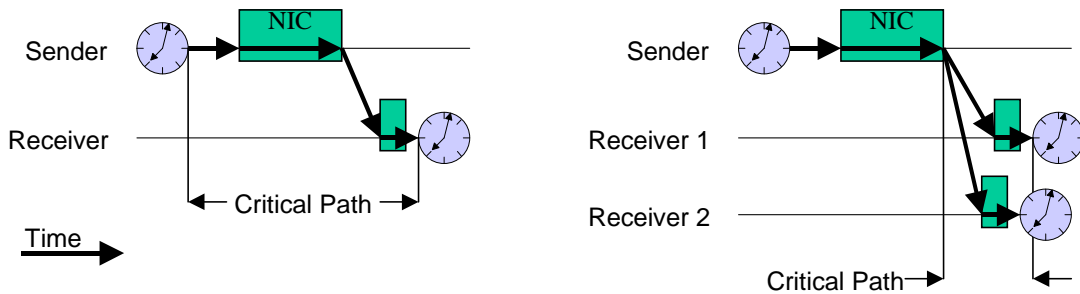


Figure 1: A critical path analysis for traditional time synchronization protocols (*left*) and RBS (*right*). For traditional protocols working on a LAN, the largest contributions to nondeterministic latency are the Send Time (from the sender’s clock read to delivery of the packet to its NIC, including protocol processing) and Access Time (the delay in the NIC until the channel becomes free). The Receive Time tends to be much smaller than the Send Time because the clock can be read at interrupt time, before protocol processing. In RBS, the critical path length is shortened to include only the time from the injection of the packet into the channel to the last clock read.

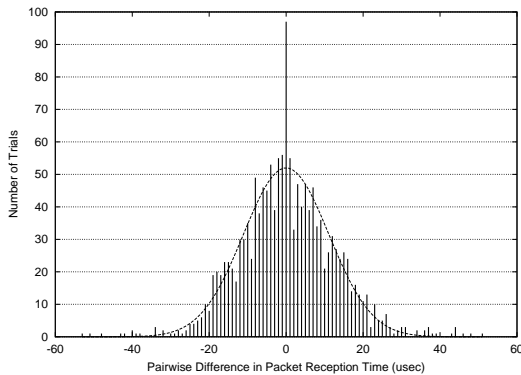


Figure 2: A histogram showing the distribution of inter-receiver phase offsets recorded for 1,478 broadcast packets, grouped into $1\mu\text{sec}$ buckets. The curve is a plot of the best-fit Gaussian parameters. ($\mu = 0, \sigma = 11.1\mu\text{sec}$, confidence=99.8%)

the course of 3 minutes, with random inter-packet delays ranging from 200msec to 2 seconds. For each pulse transmitted, we computed the difference in the pulse’s packet reception time for each of the 10 possible pairings of the 5 receivers. Some pulses were lost at some receivers due to the normal vagaries of RF links. A total of 1,478 pairings were computed.

A histogram showing the distribution of the receivers’ phase offsets is shown in Figure 2. The maximum phase error observed in any trial was $53.4\mu\text{sec}$. The Mote’s radios operate at 19,200 baud and thus have a bit time of $52\mu\text{sec}$. This correspondence supports our reasoning that a receiver’s jitter is unlikely to be much greater than a single bit time, as long as the receiver electronics have a deterministic delay between packet reception and host

notification.

The phase offset distribution appears Gaussian; the chi-squared test indicates 99.8% confidence in the best fit parameters— $\mu = 0, \sigma = 11.1\mu\text{sec}$. This is useful for several reasons. As we will see in later sections, a well-behaved distribution enables us to significantly improve on the error bound statistically, by sending multiple broadcast packets. In addition, Gaussian receivers are easy to model realistically in numeric analyses. This facilitates quick experimentation with many different algorithms or scenarios whose systematic exploration is impractical, such as examining the effects of scale.

4 Reference-Broadcast Synchronization

So far, we characterized the determinism of our receiver hardware. We now turn to the question of using that receiver to achieve the best possible clock synchronization.

We should note that our research does not encompass the question of how to build a more deterministic receiver. This is best answered by those who design them. Plans for forthcoming revisions of the Mote’s TinyOS are reported to include better phase lock between a receiver and incoming message. Instead, we ask: How well can we synchronize clocks with a *given* receiver? How is the precision we can achieve related to that receiver’s jitter?

In this section, we build up the basic RBS algorithm for nodes within a single broadcast domain. (Applying RBS in multihop networks will be considered in Section 5.)

First, we consider how RBS can be used to estimate the relative phase offsets among a group of receivers. Next, we describe a slightly more complex scheme that also corrects for clock skew. We also describe implementation of RBS on both Berkeley Motes and commodity hardware, and quantify the performance of RBS vs. NTP. Finally, we describe how the combination may be used to achieve *post-facto* synchronization.

4.1 Estimation of Phase Offset

The simplest form of RBS is the broadcast of a single pulse to two receivers, allowing them to estimate their relative phase offsets. That is:

1. A transmitter broadcasts a reference packet to two receivers (i and j).
2. Each receiver records the time that the reference was received, according to its local clock.
3. The receivers exchange their observations.

Based on this single broadcast alone, the receivers have sufficient information to form a local (or *relative*) timescale. Global (or *absolute*) timescales such as UTC are important, and we will see in Section 5.4 how RBS can be extended to provide them. However, in this section we will first consider construction of local timescales—which are often sufficient for sensor network applications. For example, if node i at position $(0, 0)$ detects a target at time $t = 4$, and j at position $(0, 10)$ detects it at $t = 5$, we can conclude that the target is moving north at 10 units per second. This comparison does not require absolute time synchronization; the reference broadcast makes it possible to compare the time of an event observed by i with one observed by j .³

The performance of this scheme is closely related to three factors:

1. The number of receivers that we wish to synchronize (n). We assumed above that $n = 2$; however, collaborative applications often require synchronization of many nodes simultaneously. As the set grows, the likelihood increases that at least one will be poorly synchronized. We define the *group*

³Our prototype implementation never sets the local clock based on reference broadcasts; instead, it provides a library function for timestamp conversion. That is, it can convert a time value that was generated by the local clock to the value that would have been generated on another node's clock at the same time, and vice-versa.

dispersion as the maximum phase error between any of the $\binom{n}{2}$ pairs of receivers in a group.

2. The nondeterminism of the underlying receiver, as we discussed in Section 3.2.
3. Clock skew in the receivers, as their oscillators all tick at different rates. A single reference will provide only *instantaneous* synchronization. Immediately afterward, the synchronization will degrade as the clocks drift apart. Precision will therefore decay as more time elapses between the synchronization pulse and the event of interest. Typical crystal oscillators are accurate on the order of one part in 10^4 to 10^6 [33]—that is, two nodes' clocks will drift 1–100 μ sec per second.

We will see in the next section how clock skew can be estimated. However, assume for the moment that we already know the clock skew and have corrected for it. Let us instead consider what is required to correct for the nondeterminism in the receiver. Recall from Figure 2 that we empirically observed the Mote's receiver error to be Gaussian. We can therefore increase the precision of synchronization statistically, by sending more than one reference:

1. A transmitter broadcasts m reference packets.
2. Each of the n receivers records the time that the reference was observed, according to its local clock.
3. The receivers exchange their observations.
4. Each receiver i can compute its phase offset to any other receiver j as the average of the phase offsets implied by each pulse received by both nodes i and j . That is, given

n : the number of receivers

m : the number of reference broadcasts, and

$T_{r,b}$: r 's clock when it received broadcast b ,

$$\forall i \in n, j \in n: \text{Offset}[i, j] = \frac{1}{m} \sum_{k=1}^m (T_{j,k} - T_{i,k}). \quad (1)$$

Because this basic scheme does not yet account for clock skew (a feature added in §4.2), it is not yet implementable on real hardware. We therefore turned to a numeric simulation based on the receiver model that we derived empirically in Section 3.2. In each trial, n nodes were given random clock offsets. m pulse transmission times were selected at random. Each pulse was “delivered” to every receiver by timestamping it using

the receiver’s clock, including a random error matching the Gaussian distribution parameters shown in Figure 2 ($\sigma = 11.1$). An offset matrix was computed as given in Equation 1. Each of these $O(n^2)$ computed offsets was then compared with the “real” offsets; the maximum difference was considered to be the *group dispersion*. We performed this analysis for values of $m = 1 \dots 50$ and $n = 2 \dots 20$. At each value of m and n , we performed 1,000 trials and calculated the mean group dispersion, and standard deviation from the mean. The results are shown in Figure 3.

This numeric simulation suggests that in the simplest case of 2 receivers (the lower curve of Figure 3a), 30 reference broadcasts can improve the precision from $11\mu\text{sec}$ to $1.6\mu\text{sec}$, after which we reach a point of diminishing return.⁴ In a group of 20 receivers, the dispersion can be reduced to $5.6\mu\text{sec}$. Figure 3b shows a 3D view of the same dataset for all receiver group sizes from 2 to 20. This view also shows that larger numbers of broadcasts significantly mitigate the effect of larger group size on precision decay.

4.2 Estimation of Clock Skew

So far, we have described a method for estimating phase offset assuming that there was no clock skew—that is, that all the nodes’ clocks are running at the same rate. Of course, this is not a realistic assumption. A complete description of crystal oscillator behavior is beyond the scope of this paper;⁵ however, to a first approximation, the important characteristics of an oscillator are:

- **Accuracy**—the agreement between an oscillator’s expected and actual frequencies. The difference is the *frequency error*; its maximum is usually specified by the manufacturer. The crystal oscillators found in most consumer electronics are accurate to one part in 10^4 to 10^6 .
- **Stability**—An oscillator’s tendency to stay at the same frequency over time. Frequency stability can be further subdivided into *short-term* and *long-term* frequency stability. Short-term instability is primarily due to environmental factors, such as variations in temperature, supply voltage, and shock. Long-term instability results from more subtle effects, such as oscillator aging.

⁴ $11\mu\text{sec}$ is the expected average result for a single pulse delivered to two receivers; this is the standard deviation of the inter-receiver phase error, found in Section 3.2, upon which the analysis was based.

⁵[33] has a good introduction to the topic and pointers to a more comprehensive bibliography.

The phase difference between two nodes’ clocks will change over time due to frequency differences in the oscillators. The basic algorithm that we described earlier is not physically realizable without an extension that takes this into account—in the time needed to observe 30 pulses, the phase error between the clocks will have changed.

Complex disciplines exist that can lock an oscillator’s phase and frequency to an external standard [26]. However, we selected a very simple yet effective algorithm to correct skew. Instead of *averaging* the phase offsets from multiple observations, as shown in Equation 1, we perform a *least-squares linear regression*. This offers a fast, closed-form method for finding the best fit line through the phase error 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.

Of course, fitting a *line* to these data implicitly assumes that the frequency is stable, i.e., that the phase error is changing at a constant rate. As we mentioned earlier, the frequency of real oscillators will change over time due, for example, to environmental effects. In general, network time synchronization algorithms (e.g., NTP) correct a clock’s phase and its oscillator’s frequency error, but do not try to model its frequency instability. That is, frequency adjustments are made continuously based on a recent window of observations relating the local oscillator to a reference. Our prototype RBS system is similar; it models oscillators as having high short-term frequency stability by ignoring data that is more than a few minutes old.

4.3 Implementation on Berkeley Motes

To test these algorithms, we first built a prototype RBS system based around the Berkeley Motes we described in Section 3.2. In the first configuration we tested, 5 Motes periodically broadcasted a reference pulse with a sequence number. Each of them used a $2\mu\text{sec}$ resolution clock to timestamp the reception times of incoming broadcasts. We then performed an off-line analysis of the data. Figure 4a shows the results of a typical experiment after automatic rejection of outliers (described below). Each point is the difference between the times that the two nodes reported receiving a reference broadcast, plotted on a timescale defined by one node’s clock. That is, given

$T_{r,b}$: r ’s clock when it received broadcast b ,

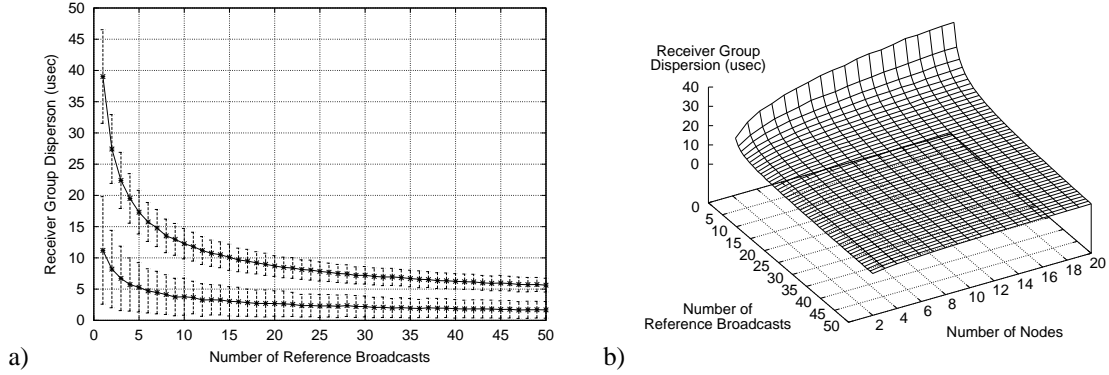


Figure 3: Analysis of expected group dispersion (i.e., maximum pairwise error) after reference-broadcast synchronization. Each point represents the average of 1,000 simulated trials. The underlying receiver is assumed to report packet arrivals with error conforming to the distribution in Figure 2. *a)* Mean group dispersion, and standard deviation from the mean, for a 2-receiver (*bottom*) and 20-receiver (*top*) group. *b)* A 3D view of the same dataset, from 2 to 20 receivers (inclusive).

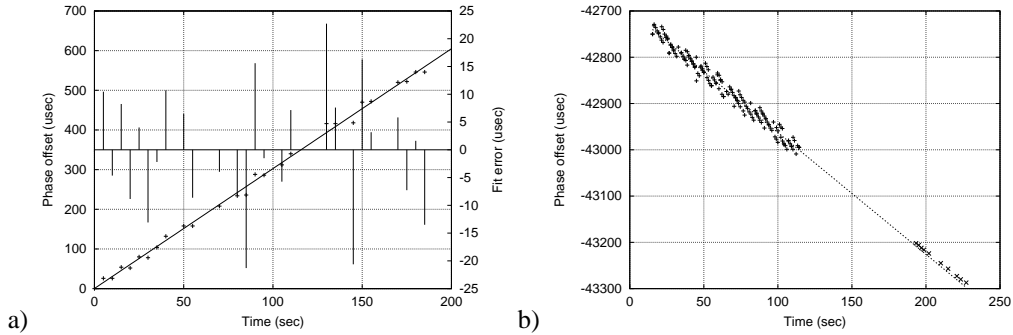


Figure 4: An analysis of clock skew’s effect on RBS. Each point represents the phase offset between two nodes as implied by the value of their clocks after receiving a reference broadcast. A node can compute a least-squared-error fit to these observations (*diagonal lines*), and thus convert time values between its own clock and that of its peer. *a)* Synchronization of the Mote’s internal clock. The vertical impulses (read with respect to the y_2 axis) show the distance of each point from the best-fit line. *b)* Synchronization of clocks on PC104-compatible single-board-computers using a Mote as NIC. The points near $x = 200$ are reference pulses, which show a synchronization error of $7.4\mu\text{sec}$ 60 seconds after the last sync pulse.

for each pulse k that was received by receivers r_1 and r_2 , we plot

$$x = T_{r_1,k}$$

$$y = T_{r_2,k} - T_{r_1,k}$$

For visualization purposes, the data are normalized so that the first pulse occurs at $(0,0)$. The diagonal line drawn through the points represents the best linear fit to the data—i.e., the line that minimizes the 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.

The residual error—that is, the RMS distance of each

point to the fit line—gives us a good idea of the quality of the fit. We can reject outliers by discarding the point that contributes most to the total error, if its error is greater than some threshold (e.g. 3σ). In this experiment, the RMS error was $11.2\mu\text{sec}$. (A more systematic study of RBS synchronization error is described in later sections.)

The slope of the best-fit line defines the clock skew, when measured by receiver r_1 ’s clock. We can see, for example, that these two oscillators drifted about $300\mu\text{sec}$ after 100 “ r_1 seconds.” The line’s intercept defines the initial phase offset. When combined, we have sufficient information to convert any time value generated by r_1 ’s clock to the value that would have been generated by

r_2 's clock at the same time. We will see in Section 5.4 that “ r_2 ” may also be an external standard such as UTC. However, even the internally consistent mapping is useful in a sensor network, as we saw in Section 4.1.

In a second test configuration, we used Motes as “network interfaces” rather than as stand-alone nodes. Motes were connected via a 9600-baud serial link to PC104-based single-board-computers running the Linux operating system. In some sense, this makes the problem more difficult because we are extending the path between the receiver (the Mote) and the clock to be synchronized (under Linux). We modified the Linux kernel to timestamp serial-port interrupts, allowing us to accurately associate complete over-the-air messages with kernel timestamps.

In this second test, shown in Figure 4b, one “Mote-NIC” sent reference broadcasts to two receivers for two minutes. Each reception was timestamped by the Linux kernel. After one minute of silence, we injected 10 reference pulses into the PC's parallel port; these pulses appear at the right of the figure. We computed the best-fit line based on the Mote-NIC pulses, after automatic outlier rejection. There was a $7.4\mu\text{sec}$ residual error between the estimate and the reference pulses.

4.4 Commodity Hardware Implementation

The performance of RBS on Berkeley Motes was very encouraging. However, it is reasonable to wonder if its success was due to the algorithm itself or simply the fact that it was implemented on a tightly integrated radio and processor platform. In addition, we were curious about the relative performance of RBS and NTP. To answer these questions—and provide RBS on a platform more generally available to users—we implemented RBS as a UNIX daemon, using UDP datagrams instead of Motes. We used a commodity hardware platform that is part of our testbed: StrongARM-based Compaq IPAQs with Lucent Technologies 11Mbit 802.11 wireless Ethernet adapters. Our IPAQs run the “Familiar” Linux distribution⁶ with Linux kernel v2.4. In this test, all the wireless Ethernet adapters are connected to a wireless 802.11 base station.

To make the comparison as fair as possible, we first implemented RBS under the same constraints as NTP: a pure userspace application with no special kernel or hardware support, or special knowledge of the MAC layer (other than that it supports broadcasts). Like NTP,

our Linux RBS daemon uses UDP datagrams sent and received via the Berkeley socket interface. Packet reception times are recorded in userspace by using the standard `gettimeofday()` library function (and underlying system call). The daemon records the time after learning that a new packet has arrived via an unblocked `select()` call. The IPAQ's clock resolution is $1\mu\text{sec}$.

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 `timevals` from one node ID to another.

The clock conversion parameters are the slope and intercept of the least-square linear regression, similar to the examples shown in Figure 4. Each regression is based on a window of the 30 most recent pulse reception reports. In practice, a small number of pulses are outliers, not within the Gaussian mode shown in Figure 2, due to random events such as concurrent serial and Ethernet interrupts. These outliers are automatically rejected based on an adaptive threshold equal to 3 times the median fit error of the set of points not yet rejected. (Early versions used a more traditional “ 3σ ” approach, but the standard deviation was found to be too sensitive to gross outliers.) The remaining points are iteratively re-fit, and the outlier threshold recomputed, until no further points are rejected. If more than half the points are rejected as outliers, the fit fails.

To test the synchronization accuracy, 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. We wrote a Linux kernel module that accepts a target pulse-time from a userspace application, then emits a GPIO pulse when the local clock indicates that the target time has arrived. By implementing the pulsing service inside the kernel, with interrupts disabled shortly before the target time arrives, there is virtually no phase error between the GPIO pulse and the node's clock. In other words, the kernel module ensures that error between the pulses as seen by the logic ana-

⁶<http://www.familiar.org>

lyzer is entirely due to the nodes' clock synchronization error, not an artifact of the test. (This kernel module purely facilitates testing; it was not used to help the synchronization of either NTP or RBS.)

Using this experimental framework, we tested three different synchronization schemes:

- RBS—Our reference broadcast synchronization. A third IPAQ was used as a pulse sender to facilitate synchronization between the two test systems. NTP was off during this test.
- NTP—The standard NTP package, version 4.1.1a, ported to our IPAQs. The `ntpd` program was first used to achieve gross synchronization. Then, the `ntpd` daemon was run, configured to send synchronization packets every 16 seconds (the maximum frequency it allows). The clients were allowed to synchronize to our lab's stratum 1 GPS-steered NTP server for several hours at this high sampling frequency before the test began. The NTP server was on the wired side of our network (i.e., accessed via the wireless base station).
- NTP-Offset—In our test, RBS has a potentially unfair advantage over NTP. Although NTP maintains a continuous estimate of the local clock's phase error with respect to its reference clock, it also limits the rate at which it is willing to correct this error. This is meant to prevent user applications from seeing large frequency excursions or discontinuities in the timescale. Our initial RBS implementation has no such restriction. Since our test only evaluates phase error, and does not give any weight to frequency stability, it might favor RBS. To eliminate this possible advantage, we created *NTP-Offset* synchronization. This was similar to the NTP method; however, during each trial, the test application queried the NTP daemon (using the `ntpq` program) for its current estimate of the local clock's phase error. This value was subtracted from the test pulse time.

For each of these synchronization schemes, we tested two different traffic scenarios:

- Light network load—The 802.11 network had very little background traffic other than the minimal load generated by the synchronization scheme itself.
- Heavy network load—Two additional IPAQs were configured as traffic generators. Each IPAQ sent

a series of randomly-sized UDP datagrams, each picked uniformly between 500 and 15,000 bytes (IP fragmentation being required for larger packets). The inter-packet delay was 10msec. The cross-traffic was entirely among third parties—that is, the source and destination of this traffic were neither the synchronization servers nor the systems under test. The average aggregate offered load of this cross-traffic was approximately 6.5Mbit/sec.

Each of the six test scenarios consisted of 300 trials, with an 8 second delay between each trial, for a total test time of 40 minutes per scenario. The results are shown in Figure 5. In the light traffic scenario, RBS performed more than 8 times better than NTP—an average of $6.29 \pm 6.45 \mu\text{sec}$ error, compared to $51.18 \pm 53.30 \mu\text{sec}$ for NTP. 95% of RBS trials had an error of $20.53 \mu\text{sec}$ or better, compared to a $131.20 \mu\text{sec}$ bound for 95% of NTP trials.

Much of the $6.29 \mu\text{sec}$ error seen with our userspace RBS implementation is due to the jitter in software—the code path through the network stack, the time required to schedule the daemon, and the system calls from userspace to determine the current time. We will see in the next section that significantly better precision can be achieved using packet timestamps acquired inside the kernel's network interrupt handler.

In our heavy traffic scenario, the difference was even more dramatic. As we discussed in Section 3, NTP is most adversely affected by path asymmetries that confound its latency estimate. When the network is under heavy load, it becomes far more likely that the medium access delay will be different during the trip to and from the NTP server. RBS, in contrast, has no dependence on the exact time packets are sent; variable MAC delays have no effect. While NTP suffered more than a 30-fold degradation in precision (average $1,542 \mu\text{sec}$, 95% within $3,889 \mu\text{sec}$), RBS was almost completely unaffected (average $8.44 \mu\text{sec}$, 95% within $28.53 \mu\text{sec}$). The slight degradation in RBS performance was due to packet loss, which reduced the number of broadcast pulses available for comparison with peers. Several instances of RBS phase excursion were also observed when a packet containing a clock parameter update was lost, forcing clients to continue using aging data.

Surprisingly, the NTP-Offset method almost uniformly performed more poorly than plain NTP. The cause was not clear, but we suspect this was due to the inter-packet jitter in the 802.11 MAC. The low-pass filter built into NTP's clock discipline algorithm was better suited to model the high-jitter network than the more responsive online phase estimator.

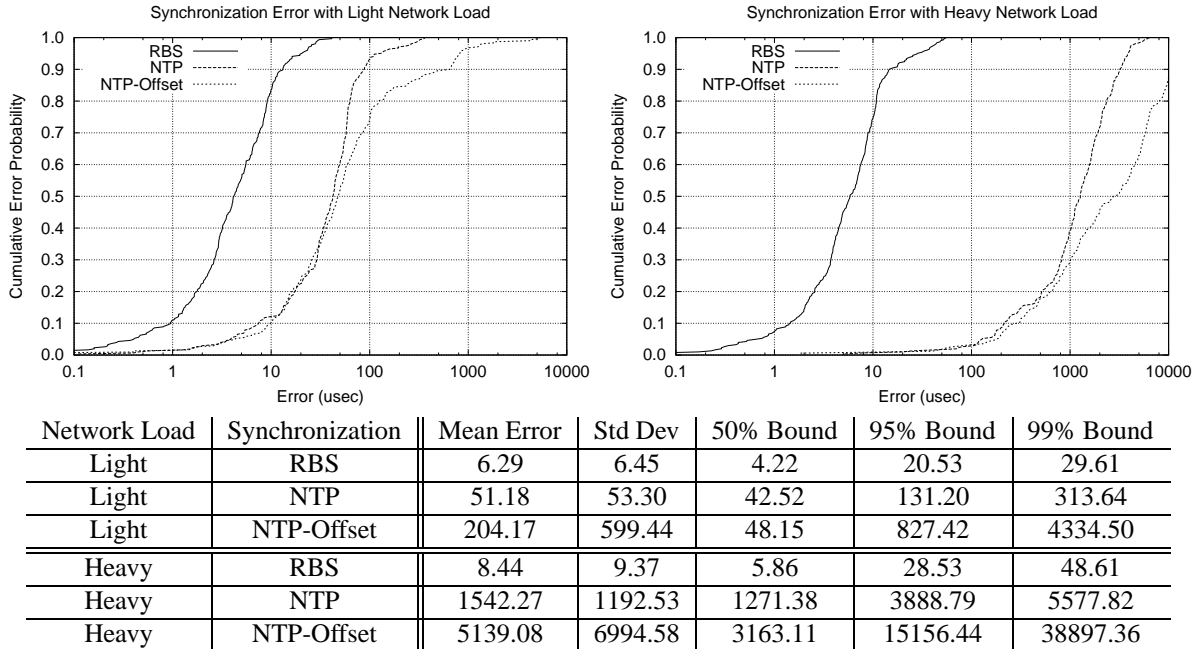


Figure 5: Synchronization error for RBS and NTP between two Compaq IPAQs using an 11Mbit 802.11 network. Clock resolution was $1\mu\text{sec}$. All units are μsec . “NTP-Offset” uses an NTP-disciplined clock with a correction based on NTP’s instantaneous estimate of its phase error; unexpectedly, this correction led to poorer synchronization. RBS performed more than 8 times better than NTP on a lightly loaded network. On a heavily loaded network, NTP further degraded by more than a factor of 30, while RBS was virtually unaffected.

4.5 RBS using 802.11 and Kernel Timestamps

In the previous section, we described the performance of RBS when implemented as a userspace application. This provided a fair comparison with NTP. However, for practical use in our testbed, we give RBS the additional advantage of packet timestamps acquired in the network interface’s interrupt handler. Timestamping at interrupt-time, before the packet is even transferred from the NIC, significantly reduces jitter and is a standard feature of the Linux kernel. The metadata is accessible by reading packets using the LBNL packet capture library, `libpcap` [21], instead of the usual socket interface.

Using kernel timestamps, the performance of RBS improved considerably, to $1.85 \pm 1.28\mu\text{sec}$ (see Figure 8), from $6.29 \pm 6.45\mu\text{sec}$ in the user-space implementation.

We believe this result was primarily limited by the $1\mu\text{sec}$ clock resolution of the IPAQ, and that finer precision can be achieved with a finer-resolution clock. In our future work, we hope to try RBS on a such a platform: for example, using the Pentium’s TSC, a counter which runs at the frequency of the processor core (e.g., 1GHz).

4.6 Application to Post-Facto Synchronization

Sensor network nodes are wireless and unattended; their finite energy becomes a dominating factor in their design. It is often not feasible to keep the processor or radio powered continuously. Such a “deep sleep” confounds traditional protocols like NTP that keep the clock synchronized at all times. We therefore have previously proposed *post-facto synchronization* [6]. In this scheme, nodes normally stay in a low power state, with *unsynchronized* clocks, until a event of interest occurs. Only after such an event are the clocks reconciled. This prevents energy from being wasted on achieving unnecessary synchronization.

An interesting facet of the RBS clock skew estimator is that it is also an effective form of post-facto synchronization. After an event of interest, nodes can power their radios up and exchange sync pulses until the best-fit line that relates nodes’ clocks to each other has been computed satisfactorily (e.g., the RMS error has fallen below some threshold). Our experiment shown in Figure 4b suggests this line can be used to precisely extrapolate backwards, estimating what the phase offset was at a time in the past, such as when the event occurred.

4.7 Summary of Advantages of RBS

So far, we have built up the basic RBS algorithm, showing how reference broadcasts can be used to relate a set of receivers' clocks to one another. Over time, we can estimate both relative phase and skew. On Berkeley Motes, RBS can synchronize clocks within $11\mu\text{sec}$, with our relatively slow 19,200 baud radios. On commodity hardware—Compaq IPAQs using an 11 Mbit/sec 802.11 network—we achieved synchronization of $6.29 \pm 6.45\mu\text{sec}$, 8 times better than NTP under the same conditions. Using kernel timestamps, precision nearly reached the clock resolution— $1.85 \pm 1.28\mu\text{sec}$.

The advantages of RBS include:

- The largest sources of nondeterministic latency can be removed from the critical path by using the broadcast channel to synchronize receivers with one another. This results in significantly better precision synchronization than algorithms that measure round-trip delay.
- Multiple broadcasts allow tighter synchronization because residual errors tend to follow well-behaved distributions. In addition, multiple broadcasts allow estimation of clock skew, and thus extrapolation of past phase offsets. This enables *post-facto synchronization*, saving energy in applications that need synchronized time infrequently and unpredictably.
- Outliers and lost packets are handled gracefully; the best fit line in Figure 4 can be drawn even if some points are missing.
- RBS allows nodes to construct *local* timescales. This is useful for sensor networks or other applications that need synchronized time but may not have an absolute time reference available. Absolute time synchronization can also be achieved, as we will describe in Section 5.4.

The primary shortcoming of RBS as we have described it thus far is that it can only be used to synchronize a set of nodes that lie within a single physical-layer broadcast domain. We address this limitation in the next section.

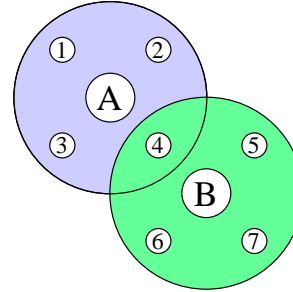


Figure 6: A simple topology where multi-hop time synchronization is required. Nodes A and B send sync pulses, establishing two locally synchronized neighborhoods. Receiver 4 hears both A and B, and can relate nodes in either neighborhood to each other.

5 Multi-Hop Time Synchronization

RBS as we have described it until now has used broadcasts to coordinate a set of receivers that lie within a single broadcast domain. A natural question that arises is how this technique might be extended so as to coordinate receivers whose span is larger than a single physical-layer broadcast. For example, the extent of wireless sensor networks is usually much larger than the broadcast range of any single node. In traditional LANs, the broadcast domain of an Ethernet is limited; LANs are interconnected with routers, bridges, or other gateways that do not propagate broadcasts at the physical layer.

In these scenarios, an extension to basic RBS can be used to synchronize a group of nodes that lie beyond the range of a single broadcast. Consider the example topology shown in Figure 6. The lettered nodes, A and B, both send a sync pulse. A and B can not hear each other, but each of them are heard by 4 receivers. Receivers that are in the same neighborhood (i.e., have heard the same sync pulse) can relate their clocks to each other, as described in previous sections. However, notice that receiver 4 is in a unique position: it can hear the sync pulses from both A and B. This allows receiver 4 to relate the clocks in one neighborhood to clocks in the other.

5.1 Multihop Clock Conversion

To illustrate how the conversion works, imagine that we wish to compare the times of two events that occurred at opposite ends of the network—near receivers 1 and 7. Nodes A and B both send synchronization pulses, at times P_A and P_B . Say that receiver 1 observes event

E_1 2 seconds after hearing A’s pulse (e.g., $E_1 = P_A + 2$). Meanwhile, receiver 7 observes event E_7 at time $P_B - 4$. These nodes could consult with receiver 4 to learn that A’s pulse occurred 10 seconds after B’s pulse: $P_A = P_B + 10$. Given this set of constraints:

$$\begin{aligned} E_1 &= P_A + 2 \\ E_7 &= P_B - 4 \\ P_A &= P_B + 10 \\ \implies E_1 &= E_7 + 16 \end{aligned}$$

In this example, it was possible to establish a global timescale for both events because there was an intersection between A’s and B’s receivers.

This technique for propagating timing information has many of the same benefits as in the single-hop case. A traditional solution to the multi-hop problem might be for nodes to re-broadcast a timing message that they have previously received. In contrast, our technique *never depends on the temporal relationship between a sender and a receiver*. As in the single-hop case, removing all the nondeterministic senders from the critical path reduces the error accumulated at each hop, as we will show in Section 5.3.

For simplicity, the example above spoke of sending “a pulse.” Naturally, the phase and skew relationship between receivers in a neighborhood can be determined more precisely by using a larger number of pulses, as we described in Section 4. To take advantage of this information, our RBS prototype does not literally compare pulse reception times, as suggested by the constraints in the equations above. Instead, it performs a series of timebase conversions using the best-fit lines that relate logically-adjacent receivers to each other. Adopting the notation $E_i(R_j)$ to mean the time of event i according to receiver j ’s clock, we can state the multihop algorithm more precisely:

1. Receivers R_1 and R_7 observe events at times $E_1(R_1)$ and $E_7(R_7)$, respectively.
2. R_4 uses A’s reference broadcasts to establish the best-fit line (as in Figure 4a) needed for converting clock values from R_1 to R_4 . This line is used to convert $E_1(R_1)$ to $E_1(R_4)$.
3. R_4 similarly uses B’s broadcasts to convert $E_1(R_4)$ to $E_1(R_7)$.
4. The time elapsed between the events is computed as $E_1(R_7) - E_7(R_7)$.

This algorithm is conceptually the same as the simpler version. However, each timebase conversion implicitly includes a correction for skew in all three nodes.

5.2 Time Routing in Multihop Networks

In Figure 6, there was a single “gateway” node that we designated for converting timestamps from one neighborhood’s timebase to another. However, in practice, no such designation is necessary, or even desirable. It is straightforward to compute a “time route”—that is, dynamically determine a series of nodes through which time can be converted to reach a desired final timebase.

Consider the network topology in Figure 7a. As in Figure 6, the lettered nodes represent sync pulse senders; numbered nodes are receivers. (This distinction between senders and receiver roles is purely illustrative; in reality, a node can play both roles.) Each pulse sender creates a neighborhood of nodes that have known phase offsets by virtue of having received pulses from a common source. These relationships are visualized in Figure 7b, in which nodes with known clock relationships are joined by graph edges. A path through this graph represents a series of timebase conversions. For example, we can compare the time of $E_1(R_1)$ with $E_{10}(R_{10})$ by transforming $E_1(R_1) \rightarrow E_1(R_4) \rightarrow E_1(R_8) \rightarrow E_1(R_{10})$.

It is possible to find a series of conversions automatically, simply by performing a shortest-path search in a graph such as in Figure 7b. In addition, the weights of the graph edges can be used to represent the quality of the conversion—for example, the residual error (RMS) of the linear fit to the broadcast observations. A minimum-error conversion between any two nodes can be found using a weighted-shortest-path search such as Dijkstra’s or Bellman-Ford.

Of course, such shortest-path algorithms do not scale to large networks due to their dependence on global information. Although there is precedent for such systems (e.g., the Internet’s early link-state routing algorithms), a more localized approach is needed if the method is to scale. To this end, there is an interesting alternative: time conversion can be built into the extant packet forwarding mechanism. That is, nodes can perform successive time conversions on packets as they are forwarded from node to node—keeping the time with respect to the local clock at each hop. This technique, also suggested by Röemer [28], is attractive because it requires only local computation and communication. Instead of flooding clock conversion parameters globally, they can be distributed using a tightly scoped broadcast. In addition, the quality

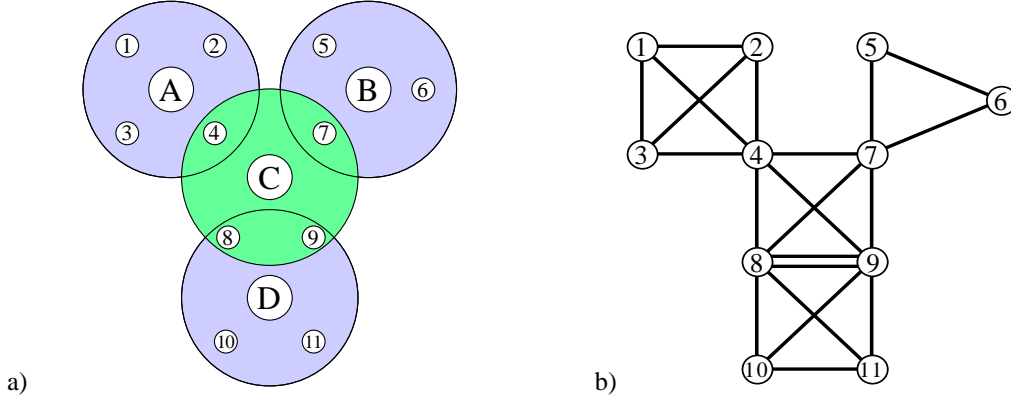


Figure 7: A more complex (3-hop) multihop network topology. *a)* The physical topology. Beacons (lettered nodes), whose transmit radii are indicated by the larger shaded areas, broadcast references to their neighbors (numbered nodes). *b)* The logical topology. Edges are drawn between nodes that have received a common broadcast, and therefore have enough information to relate their clocks to each other. Receivers 8 and 9 share two logical links because they have two receptions in common (from C and D).

of time synchronization across each link can be incorporated into the link metric used by the routing algorithm, ensuring that routing is not completely orthogonal to the quality of the time conversions available.

5.3 Measured Performance of Multihop RBS

In a multihop topology where a series of timestamp conversions are required, as described in the previous section, each conversion step can introduce synchronization error. We now consider the cumulative effects of such errors as the path length grows.

We saw in Section 3.2 that the synchronization error introduced by a reference broadcast is Gaussian. In addition, each of the per-hop contributions to the error are *independent* because the phase and skew estimates at each hop are based on a different set of broadcasts. Therefore, we expect that total path error—a sum of independent Gaussian variables—will also follow a Gaussian distribution. If the average per-hop error along the path is σ , then we expect⁷ the average path error for an n -hop path to be $\sigma\sqrt{n}$. This bound is pleasing in that it grows slowly, and implies that we can synchronize nodes across many hops without significant decay in precision.

We tested the precision of multihop RBS using the IPAQ and 802.11-based testbed we described in Section 4.4, including the in-kernel packet timestamping discussed in Section 4.5. We configured 9 IPAQs in a linear topol-

⁷From the variance’s identity that $\text{var}(x+y) = \text{var}(x) + \text{var}(y)$; $\text{variance}=\sigma^2$.

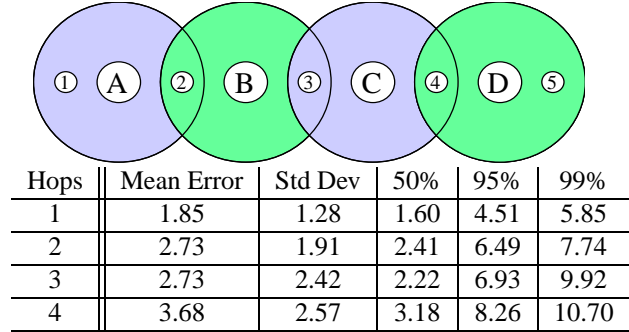


Figure 8: *top)* The topology used to test cumulative error when RBS is used in a multihop network. *bottom)* Measured performance of multihop RBS, using kernel timestamping, on IPAQs using 11 MBit/sec 802.11. All units are μsec .

ogy, alternating between 5 (numbered) receivers and 4 (lettered) beacon emitters, as shown in Figure 8. Each receiver could hear only the nearest sender on its left and right. The test procedure was the same as we described in Section 4.4—in 300 trials, two IPAQs were commanded to raise a GPIO pin high at the same time. A logic analyzer reported the actual time differences. We tested a 1-hop, 2-hop, 3-hop and 4-hop conversion, by testing the rise-time difference between nodes 1–2, 1–3, 1–4, and 1–5, respectively.

The results, shown in Figure 8, generally seem to follow our $\sigma\sqrt{n}$ estimate. The average 4-hop error was $3.68 \pm 2.57\mu\text{sec}$, which is nearly $\sigma\sqrt{4}$, normalizing σ to be the average 1-hop error.

5.4 Synchronization with External Timescales

So far, we have spoken entirely of creating local, internally consistent timescales. Although relative synchronization is sufficient for many applications, others require absolute time as measured by an external reference such as UTC. Reference timescales are typically distributed via governmentally sponsored radio systems such as the short-wave WWVB station [3] or the satellite-based Global Positioning System [16]. Commercially available receivers for these systems can synchronize computers by providing them with a “PPS” (pulse-per-second) signal at the beginning of each second. The seconds are numbered out of band, such as through a serial port.

RBS provides a natural and precise way to synchronize a network with such an external timescale. For example, consider a GPS radio receiver connected to one of the nodes in a multihop network such as the one in Figure 7. GPS time simply becomes another node in that network, attached via one edge to its host node. The PPS output of the receiver can be treated exactly as normal reference broadcasts are. That is, the host node timestamps each PPS pulse and compares its own timestamp with the pulse’s true GPS time. The phase and skew of the host node’s clock relative to GPS time can then be recovered using the algorithms described in Section 4. Other nodes can use GPS time by finding a multihop conversion route to it through the host node, using the algorithm we described in Section 5.2.

6 Application Experience

To date, our RBS implementation has been applied, both within and without our research group, in three distributed sensor network applications requiring high-precision time synchronization. Each uses RBS-synchronized clocks to precisely measure the arrival time of acoustic signals. Most audio codecs sample the channel at 48KHz, or $\approx 21\mu\text{sec}$ per sample. As we have seen in previous sections, the precision nominally achieved by RBS is significantly better. Distributed acoustic sensors have thus been able to correlate their acoustic time-series down to individual samples. This has been useful in a number of contexts.

Our group has developed a implemented a centimeter-scale localization service for Berkeley Motes based on acoustic time-of-flight ranging [8]. A set of IPAQs set

around the room first establish their own positions within a relatively coordinate system by ranging to one another. Each IPAQ emits an audible “chirp” which has an encoded pseudonoise sequence [9]. IPAQs run a matched filter over their incoming audio data to find the most likely audio sample indicating arrival of the chirp. 802.11-based RBS corrections are then applied to convert this into the equivalent sample number in the sender’s timebase, allowing time of flight and therefore range to be computed. Once this startup phase is complete, our “acoustic Motes,” specially equipped with small amplifiers and speakers, periodically emit a similar pseudonoise chirp; IPAQs in the region each compute their ranges to the Mote, and can localize it by trilaterating. In this case, RBS is used to synchronize all Motes in the system to each other. A special “MoteNIC”—a Mote physically attached to an IPAQ—provides translations between the Mote and IPAQ time domains.

Merrill *et al.* describe Sensoria Corporation’s use of RBS in their implementation of a distributed, wireless embedded system capable of autonomous position location with accuracy on the order of 10cm [22]. This system was built under the auspices of the DARPA SHM program and has been field tested at Fort Leonard Wood, Missouri, in configurations of up to 20 nodes. Their application is also notable because it represents a third hardware and radio platform on which RBS has been successfully implemented. Sensoria’s “WINS NG” node is based around the Hitachi SH4 processor running Linux 2.4. Each node has two low-power 56Kbit/sec hybrid TDMA/FHSS radios with an RS-232 serial interface to the host processor. As with our 802.11 implementation, the firmware of this radio was opaque, making schemes that rely on tricks at the MAC layer impossible.

Finally, blind beamforming on acoustic signals has been studied by Yao *et al.* for a number of years [35]. Their group had previously restricted their empirical studies to centralized systems, as they did not have a platform capable of synchronizing distributed audio sampling with sufficient precision. Recently, Wang *et al.* reported their first experience building a *distributed* beam-forming application on IPAQs using our RBS daemon [34].

7 Conclusions and Future Work

In this paper, we explored a form of time synchronization, *Reference-Broadcast Synchronization*, that provides more precise, flexible, and resource-efficient net-

work time synchronization than traditional algorithms. The fundamental property of our design is that it *synchronizes a set of receivers with one another*, as opposed to traditional protocols in which senders synchronize with receivers. In addition, we have presented a novel multi-hop algorithm that allows this fundamental property to be maintained across broadcast domains.

In our scheme, nodes periodically send beacon messages to their neighbors using the network’s physical-layer broadcast. Recipients use the message’s arrival time as a point of reference for comparing their clocks. The message contains no explicit timestamp, nor is it important exactly when it is sent.

RBS has a number of properties that make it attractive. First, by using the broadcast channel to synchronize receivers with one another, the largest sources of nondeterministic latency are removed from the critical path. This results in significantly better-precision synchronization than algorithms that measure round-trip delay. The residual error is often a well-behaved distribution (e.g., Gaussian), allowing further improvement by sending multiple reference broadcasts. The extra information produces significantly better estimates of relative phase and frequency, and allows graceful degradation in the presence of outliers and lost packets.

The RBS clock skew estimate also supports extrapolation of *past* phase offsets. This enables nodes to synchronize *post-facto*, saving energy in applications that need synchronized time infrequently and unpredictably.

We presented a quantitative study that compared an RBS implementation to a carefully tuned installation of GPS-steered NTP. Both used IPAQ PDAs with 802.11 wireless Ethernet. We found that the average synchronization error of RBS was $6.29 \pm 6.45\mu\text{sec}$, 8 times better than that of NTP in a lightly loaded network. A heavily loaded network further degraded NTP’s performance by a factor of 30 but had little effect on RBS. With kernel timestamping hints, RBS achieved an average error of $1.85 \pm 1.28\mu\text{sec}$, which we expect was limited by the IPAQ’s $1\mu\text{sec}$ clock resolution.

Our multihop scheme allows locally coordinated timescales to be federated into a global timescale, across broadcast domains. Precision decays slowly—the average error for an n -hop network is proportional to \sqrt{n} . In our test of kernel-assisted RBS across a 4-hop topology, average synchronization error was $3.68 \pm 2.57\mu\text{sec}$. RBS-federated timescales may also be referenced to an external standard such as UTC if at least one node has access to a source of reference time.

We have implemented RBS on a variety of hardware platforms, where it has proven to be robust and reliable for both performance measurement and in support of real applications. However, there is still much work to be done. Some important scaling issues have not yet been explored, such as automatic, dynamic election of the set of nodes to act as beacon senders. If there are multiple beacon-senders in a single neighborhood, RBS can make use of redundant information to improve precision; however, it quickly reaches the point of diminishing return where the system would be better off conserving bandwidth instead. We would like understand just how much extra precision is afforded by this redundancy.

In addition, we would like to more fully quantify a number of performance questions: precision vs. both the bandwidth used for beacons, and their frequency; minimum time to acquire synchronization to within a given precision bound; post-facto synchronization precision vs. time elapsed from event to reference pulses; and precision with different data filtering algorithms. We would like to quantify the performance of RBS when used for external (UTC) synchronization over multiple hops, vs. using NTP in the same topology.

We are confident that these techniques are widely applicable, based on our experience with Berkeley Motes running TinyOS, Linux-based IPAQs, PCs, and Sensoria’s WinsNG radios. Each has quirks that have taught us important lessons about RBS, so we would like further with experience with RBS with a wider range of hardware platforms, network interfaces, operating environments, and applications. Many of our collaborators in the sensor network community have research interests that require precise time synchronization: collaborative signal processing, acoustic ranging, object tracking, coordinated actuation, and so forth. We look forward to evaluating the utility of RBS in support of these applications.

Acknowledgments

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. The authors are indebted to Sensoria Corporation for support and feedback. Members of the UCLA LECS laboratory, the anonymous reviewers, and OSDI shepherd Eric Brewer provided insightful comments that significantly improved our research.

References

- [1] I. F. Akyildiz, W. Su, Y. Sankarasubramanian, and E. Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38(4):393–422, March 2002.
- [2] G. Asada, M. Dong, T.S. Lin, F. Newberg, G. Pottie, W.J. Kaiser, and H.O. Marcy. Wireless Integrated Network Sensors: Low Power Systems on a Chip. In *Proceedings of the European Solid State Circuits Conference*, 1998.
- [3] R.E. Beehler. Time/frequency services of the U.S. National Bureau of Standards and some alternatives for future improvement. *Journal of Electronics and Telecommunications Engineers*, 27:389–402, Jan 1981.
- [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] Flaviu Cristian. Probabilistic clock synchronization. *Distributed Computing*, 3:146–158, 1989.
- [6] Jeremy Elson and Deborah Estrin. Time synchronization for wireless sensor networks. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS-01)*. IEEE Computer Society, April 23–27 2001.
- [7] Saurabh Ganeriwal, Ram Kumar, Sachin Adlakha, and Mani B. Srivastava. Network-wide Time Synchronization in Sensor Networks. Technical report, University of California, Dept. of Electrical Engineering, 2002.
- [8] Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah Estrin. Locating tiny sensors in time and space: A case study. In *In Proceedings of the International Conference on Computer Design (ICCD 2002)*, Freiburg, Germany, September 2002. <http://lecs.cs.ucla.edu/Publications>.
- [9] Lewis Girod and Deborah Estrin. Robust range estimation using acoustic and multimodal sensing. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2001)*, March 2001.
- [10] R. Gusell and S. Zatti. The accuracy of clock synchronization achieved by TEMPO in Berkeley UNIX 4.3 BSD. *IEEE Transactions on Software Engineering*, 15:847–853, 1989.
- [11] Jason Hill and David Culler. A wireless embedded sensor architecture for system-level optimization. Technical report, U.C. Berkeley, 2001.
- [12] Jason Hill, Robert Szwedczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.
- [13] 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.
- [14] ISO/IEC. IEEE 802.11 Standard. IEEE Standard for Information Technology, ISO/IEC 8802-11:1999(E), 1999.
- [15] 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 international conference on Mobile computing and networking*, pages 271–278, 1999.
- [16] Elliott D. Kaplan, editor. *Understanding GPS: Principles and Applications*. Artech House, 1996.
- [17] H. Kopetz and W. Schwabl. Global time in distributed real-time systems. Technical Report 15/89, Technische Universität Wien, 1989.
- [18] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–65, 1978.
- [19] Cheng Liao, Margaret Martonosi, and Douglas W. Clark. Experience with an adaptive globally-synchronizing clock algorithm. In *Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '99)*, pages 106–114, New York, June 1999.
- [20] J. Mannermaa, K. Kalliomaki, T. Mansten, and S. Turunen. Timing performance of various GPS receivers. In *Proceedings of the 1999 Joint Meeting of the European Frequency and Time Forum and the IEEE International Frequency Control Symposium*, pages 287–290, April 1999.
- [21] 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.
- [22] William Merrill, Lewis Girod, Jeremy Elson, Kathy Sohrobi, Fredric Newberg, and William Kaiser. Autonomous Position Location in Distributed, Embedded, Wireless Systems. In *Proceedings of IEEE CAS Workshop on Wireless Communications and Networking*, Pasadena, CA, September 2002. <http://www.sensoria.com/publications.html>.
- [23] R. M. Metcalfe and D. R. Boggs. Ethernet: Distributed packet switching for local computer networks. *Communications of the ACM*, 26(1):90–95, January 1983.
- [24] 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.
- [25] David L. Mills. Precision synchronization of computer network clocks. *ACM Computer Comm. Review*, 24(2):28–43, April 1994.
- [26] David L. Mills. Adaptive hybrid clock discipline algorithm for the network time protocol. *IEEE/ACM Transactions on Networking*, 6(5):505–514, October 1998.
- [27] M. Mock, R. Frings, E. Nett, and S. Trikaliotis. Continuous clock synchronization in wireless real-time applications. In *The 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*, pages 125–133, Washington - Brussels - Tokyo, October 2000. IEEE.
- [28] Kay Römer. Time synchronization in ad hoc networks. In *Proceedings of MobicHoc 2001*, Long Beach, CA, Oct 2001.
- [29] I. Rubin. Message Delays in FDMA and TDMA Communication Channels. *IEEE Trans. Communin.*, COM27(5):769–777, May 1979.
- [30] T. K. Srikanth and Sam Toueg. Optimal clock synchronization. *J-ACM*, 34(3):626–645, July 1987.
- [31] Paulo Veríssimo and Luis Rodrigues. A posteriori agreement for fault-tolerant clock synchronization on broadcast networks. In Dhiraj K. Pradhan, editor, *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing (FTCS '92)*, page 85, Boston, MA, July 1992. IEEE Computer Society Press.
- [32] Paulo Veríssimo, Luis Rodrigues, and Antonio Casimiro. Cesium-spray: a precise and accurate global time service for large-scale systems. Tech. Rep. NAV-TR-97-0001, Universidade de Lisboa, 1997.
- [33] John R. Vig. Introduction to Quartz Frequency Standards. Technical Report SLCET-TR-92-1, Army Research Laboratory, Electronics and Power Sources Directorate, October 1992. Available at <http://www.ieee-uffc.org/freqcontrol/quartz/vig/vigtoc.htm>.
- [34] 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. <http://lecs.cs.ucla.edu/Publications>.
- [35] 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.