

# Locating tiny sensors in time and space: A case study

Lewis Girod, Vladimir Bychkovskiy, Jeremy Elson, and Deborah Estrin  
 Department of Computer Science  
 University of California, Los Angeles USA 90095  
 {girod,vladimir,jelson,destrin}@lecs.cs.ucla.edu

**Abstract**—As the cost of embedded sensors and actuators drops, new applications will arise that exploit high density networks of small devices capable of a variety of sensing tasks. Although individual devices may have limited functionality, the true value of the system comes from the emergent behavior that arises when data from many places in the system is combined. This type of data fusion has a number of requirements, but two of the most important are: 1) synchronized time, precise enough to resolve movement in the sensed phenomenon (e.g., sound); and 2) known geographic locations, on a similar scale to the sensors’ size and deployment density. However, the installation cost of a localization system with sufficient granularity is considerable, because of the large amount of effort required to deploy such a system and make all the measurements required to tune it. In this paper, we describe a system based on COTS components that incorporates our novel time synchronization and acoustic ranging techniques. The result is a low-cost, readily available platform for distributed, coherent signal processing.

## I. 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]. The vision of many researchers is to create sensor-rich “smart environments” through planned or ad-hoc deployment of thousands of sensors, each with a short-range wireless communications channel, and capable of detecting ambient conditions such as temperature, movement, sound, light, or the presence of certain objects.

In almost any distributed sensor application, there are two critical pieces of infrastructure:

- 1) *Time synchronization*, precise enough to resolve significant movement in the sensed phenomenon; and
- 2) *Spatial localization*, with resolution similar to the nodes’ size and deployment density.

There are many examples of sensor network tasks that require both synchronized time and known sensor locations: for example, to integrate a time-series of proximity detections into a velocity estimate [3]; to measure the time-of-flight of sound for localizing its source [6]; to distribute a beamforming array [13]; or to suppress redundant messages by recognizing that they describe duplicate detections of the same event by different but nearby sensors [9].

While spatial localization of nodes is possible to implement manually—by carefully measuring and “hard-coding” the locations of the sensors—systems that discover location autonomously are advantageous for reasons of convenience, cost, and application to ad-hoc deployment scenarios.

In this paper, we discuss the development of a system, based on commercial off-the-shelf (COTS) components, which is capable of automatic localization and time synchronization with sufficient precision (on the order of 10cm and 10 $\mu$ sec) to support distributed, coherent signal processing. Our system’s time synchronization is an implementation of Reference-Broadcast Synchronization (RBS), described more fully in [4]. Localization is based on an underlying ranging system that works by timing the flight of a wideband acoustic pulse, described in [5].

The remainder of this paper is organized as follows. In Section II, we describe the hardware platforms that compose our testbed. We give an overview of the software components of our system in Section III. A more detailed description of the subsystems is found in Section IV (time synchronization) and Section V (acoustic ranging and localization). Finally, in Section VI, we describe our conclusions and future work.

## II. HARDWARE PLATFORMS

Although Moore’s law predicts that hardware for sensor networks will inexorably become smaller, cheaper, and more powerful, technological advances will never prevent the need to make tradeoffs. Even as our notions of metrics such as “fast” and “small” evolve, there will always be compromises: nodes will need to be faster *or* more energy-efficient, smaller *or* more capable, cheaper *or* more durable.

Instead of choosing a single hardware platform that makes a particular set of compromises, we believe an effective design is one that uses a tiered platform consisting of a heterogeneous collection of hardware. Small, cheap, and computationally limited nodes (“motes”, after the Berkeley Smart Dust project [10]) can be used more effectively by augmenting the network with larger, faster, and more expensive hardware (“bases”). An analogy can be made to the memory hierarchy commonly found in desktop computer systems. CPUs typically have an expensive but fast on-chip cache, backed by slower but larger L2 cache, main memory, and ultimately on-disk swap space. This organization, combined with a tendency in computation for locality of reference, results in a memory system that appears to be as large and as cheap (per-byte) as the swap space, but as fast as the on-chip cache memory. In sensor networks, where localized algorithms are a primary design goal [9], similar benefits can be realized by creating the network from a spectrum of hardware ranging from small, cheap, and numerous, to large, expensive, and powerful.

### A. Motes

The smallest nodes in our testbed are the “COTS Mote,” originally developed at U.C. Berkeley [10], [7] and now commercially available from Crossbow Technologies<sup>1</sup>. It is equipped with a low-power, 4MHz, 16-bit microprocessor (Atmel 90LS8535), with 8K of program memory and 512 bytes of SRAM. An integrated RF Monolithics 916.50 MHz transceiver (TR1000) provides narrowband wireless connectivity at 19.2Kbps. Communication with a directly attached host is possible via a serial port. A number of GPIO pins, A/D, and D/A converters are also available for I/O purposes.

The original COTS Mote is pictured in Figure 1. Different versions of the basic design have various combinations of sensors: temperature, light, humidity, acceleration, and so forth. In our localization testbed, we use a locally developed variation of the mote, the “acoustic mote,” on which one of the D/A converter outputs is attached to an audio amplifier circuit and speaker.

<sup>1</sup><http://www.xbow.com>

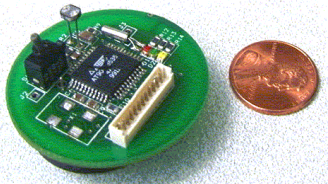


Fig. 1. Pister’s “COTS Mote”, developed at U.C. Berkeley

The motes’ operating environment is TinyOS [8], [7], a small, event-based operating system developed specifically for the mote by Hill, et.al. at U.C. Berkeley.

### B. Bases

Although the motes have advantages (small, cheap, low-power, long-lived), they have very limited capabilities. For example, it is impossible to do anything more than the most basic forms signal processing: the mote is too slow to process a time series in real-time, and doesn’t have enough memory to buffer more than a few dozen samples for offline analysis. We have found that it is possible to implement much richer applications if there are a few larger, more computationally endowed nodes available in the network as well.

Our bases are Compaq iPAQ 3760s, which are handheld, battery-powered devices normally meant to be used as PDAs. We selected the iPAQ because it has reasonable battery life, has the peripherals needed by our project, supports Linux, is readily available, and is usable right off the shelf.<sup>2</sup> The iPAQ has a 137MHz Intel StrongARM-1110 processor, 32MB of RAM and 32MB of FLASH for persistent storage. The standard model also comes equipped with a built-in speaker and microphone which we use for acoustic ranging, as we will see in Section V. 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), making them capable of communicating at higher bandwidth, longer range, and with greater reliability than the motes.

Our testbed’s iPAQs use the StrongARM port of the Linux operating system (the “Familiar” distribution [1]). This combination of hardware and operating system provides a powerful and convenient development environment similar to a standard desktop operating system.

## III. SYSTEM OVERVIEW

Ultimately, the goal of our system is to first establish a coordinate system defined by the positions of the iPAQs at startup, then continuously monitor the position of the motes within this coordinate system. We assume the iPAQs are initially placed around a room “at random” but then remain in fixed locations, or move rarely. The iPAQs form an ad-hoc infrastructure, while the smaller motes are assumed to be constantly in motion—perhaps attached to people or objects that are being tracked as they move through the environment.

The functional components of the localization system are shown in Figure 2. Localization is based on an acoustic ranging system that uses a wideband pseudonoise sequence to measure the time of flight of sound from one point to another (*bottom panel*). Ranging is first used to determine iPAQ-to-iPAQ distances and federate an iPAQ coordinate system; this mode uses both the iPAQ’s speaker and microphone (*center panel*). Once the coordinate system has been established, ranging

<sup>2</sup>Previous incarnations of our testbed [3] were based on hardware platforms that we integrated ourselves from components; we have since learned not to underestimate the value of a platform that comes with its own integrated power supply, enclosure, I/O, etc.

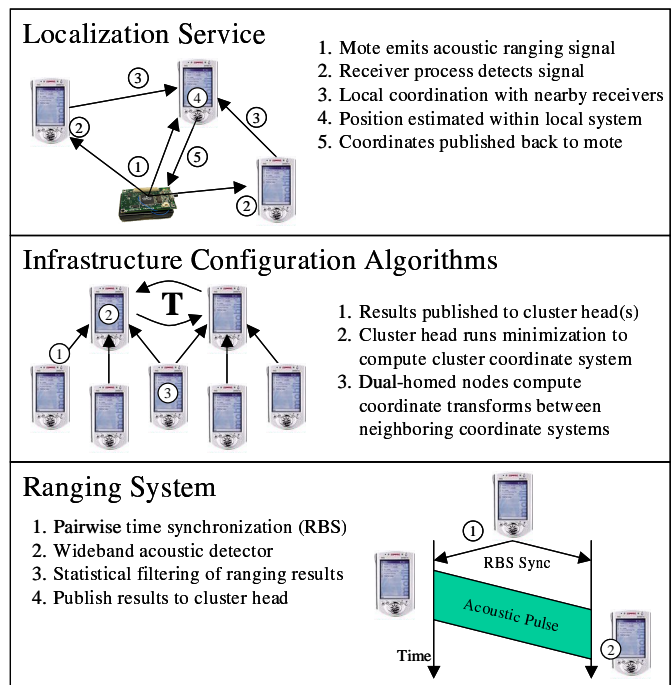


Fig. 2. High-level System Diagram. At the lowest layer, a ranging system generates pairwise range values. These ranges are propagated to a cluster head that runs an optimization algorithm to construct a consistent coordinate system and reject the many forms of error in the source data. Once a coordinate system has been computed, the system can localize devices in the region.

from an acoustic mote’s emitter to the iPAQs’ microphones is used to localize motes within this coordinate system (*top panel*).

Many existing acoustic ranging systems use a somewhat ad-hoc time synchronization strategy that assumes there is a tight coupling between the acoustic and RF components in the system. Those systems are designed to generate an acoustic pulse and an RF synchronization packet at exactly the same time. We have found this to be a poor design choice for a number of reasons—for example, the system tends to be more complex due to tight inter-module dependencies, and ranging experiments are vulnerable to individual lost packets. In contrast, our system contains a separate module that continuously maintains timebase conversion metrics among all the components in the system, providing this information to the ranging system when necessary. By abstracting this part of the system away, the overall design is simplified and the system becomes much more robust to packet loss. In addition, the precision of the synchronization is improved: by averaging over many packet observations, outliers can be rejected and clock skew corrections are possible.

We mentioned in the previous section that motes use narrowband 900MHz radios to communicate with each other, while the iPAQs use 802.11b wireless Ethernet cards. To bridge the gap between these two domains, several of the iPAQs are configured with a “MoteNIC”—that is, a mote attached to the iPAQ via the serial port. Such motes can act as a network interface visible to processes running on the iPAQ (under Linux). These iPAQs are then capable of routing information between the mote and iPAQ domains.

## IV. TIME SYNCHRONIZATION

The time synchronization module is responsible for computing parameters that relate the phase and frequency of all of the system’s clocks to one another. This is, perhaps, a more complex task than it might seem—in addition to being a distributed system, it is a hetero-

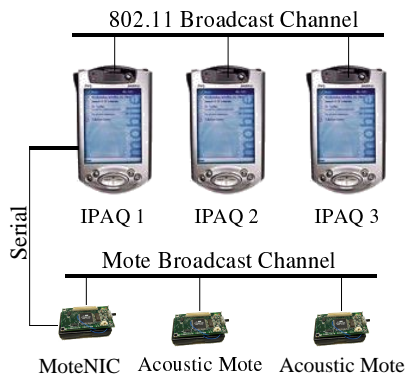


Fig. 3. Our hardware configuration, where multi-hop timesync is required to relate the acoustic mote’s timebase to the iPAQs.

geneous one. Specifically, there are three types of components in the system whose clocks run independently and must be reconciled:

- The system clock in each mote
- The system clock in each iPAQ
- Each iPAQ’s codec sample clock

The synchronization service builds up a table of conversion parameters that relate many of the different clocks in the system to each other. A series of conversions may be necessary to convert a timestamp in one timebase to a timestamp in another. For example, consider the configuration shown in Figure 3. iPAQ 1 is configured as a gateway node, i.e., with a MoteNIC as described in Section III. Imagine that we wish to measure the time of flight of a sound from an acoustic mote to iPAQ 2. This means we need to relate the timebase of the acoustic mote to the timebase of iPAQ 2’s codec sample clock. This requires a series of 4 conversions:

- 1) Network time synchronization (via mote radios) from the acoustic mote’s clock to mote clock in iPAQ 1’s MoteNIC
- 2) Intra-node synchronization from iPAQ 1’s mote clock to iPAQ 1’s system clock
- 3) Network time synchronization from iPAQ 1 to iPAQ 2 (via 802.11)
- 4) Intra-node synchronization from iPAQ 2’s system clock to iPAQ 2’s clodec sample clock

The methods for intra-node and network time synchronization will be discussed in detail below. On each iPAQ, those synchronization methods populate a table of conversion parameters between the clocks in the system, annotated with an estimate of the RMS error of each conversion. Given the parameters that make up this graph, the series of parameters that make up the minimum-error conversion between the desired source and destination timescales is automatically computed, using a weighted shortest-path search algorithm.

#### A. Network time synchronization

In our system, network time synchronization (mote-to-mote and iPAQ-to-iPAQ) is performed using an implementation of Reference-Broadcast Synchronization, or RBS, described in more detail in [4]. Briefly, the fundamental property of RBS is that it *synchronizes a set of receivers with one another*, as opposed to traditional protocols in which senders synchronize with receivers. In this way, 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), so precision of the phase offset estimate can be significantly improved by sending multiple reference broadcasts.

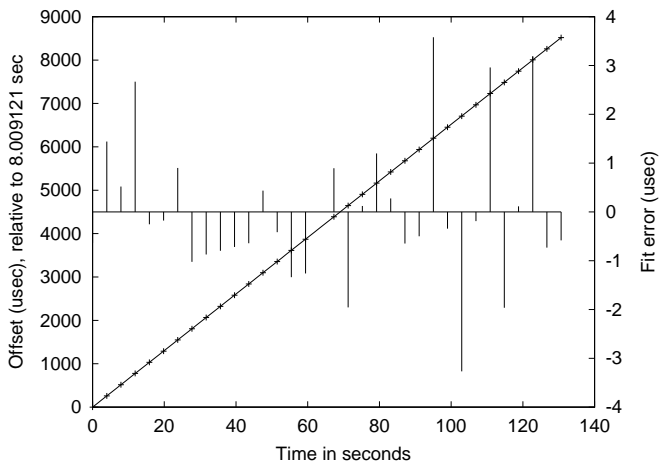


Fig. 4. Typical linear regression to recover relative phase and skew between two nodes in the system using RBS. Each point represents the phase offset between two nodes as implied by the value of their clocks after receiving a reference broadcast. A node can compute a least-squared-error fit to these observations (*diagonal line*), and thus convert time values between its own clock and that of its peer. The vertical impulses (read with respect to the  $y^2$  axis) show the distance of each point from the best-fit line.

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.)

Complex disciplines exist that can lock an oscillator’s phase and frequency to an external standard [11]. However, we selected a very simple yet effective algorithm to correct skew: a *least-squares linear regression* on the time series 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 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.

An example of such a linear regression on real data from our system is shown Figure 4. Each regression is based on a window of the 30 most recent pulse reception reports. Outliers are 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 “ $3\sigma$ ” 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.

Motes and iPAQs both participate in this scheme separately. That is, iPAQs synchronize with each other using 802.11 broadcasts, while motes synchronize with each other using their narrowband radio broadcasts. The basic scheme is the same on both motes and iPAQs, except that motes do not compute their own conversion parameters, as iPAQs do—they forward all of their broadcast reception reports to an iPAQ for post-processing instead.

In practice, iPAQ-to-iPAQ synchronization via 802.11 has an error

of  $\approx 1.5\mu\text{sec}$ ; this is limited primarily by the iPAQ's  $1\mu\text{sec}$  clock resolution. Motes use slower (19.2kbit/sec) radios with a bit time of  $53\mu\text{sec}$ ; nevertheless, RBS achieves  $10\mu\text{sec}$  error through averaging and outlier rejection. A detailed performance study of RBS can be found in [4].

Of course, motes and iPAQs can not directly observe each other's broadcasts since they use different radios (different frequencies, different MAC layers, etc.). An extra piece is needed to relate the mote timescales with the iPAQ timescales. This is where intra-node synchronization comes into play.

### B. Intra-node synchronization

As we discussed at the beginning of this section, each iPAQ can consist of three separate clocks: 1) the system clock; 2) the audio codec's sample clock, and 3) if the iPAQ has a MoteNIC attached via serial port, the mote's system clock. In addition to supporting RBS as described above, our time service daemon also supports synchronization between components within a system. The system's device drivers periodically inject pairs of time values into the time daemon, where each pair represents the value of two of the system's clocks at the same instant. This allows the clocks to be related. There are two types of intra-node synchronization in our system:

- *iPAQ to MoteNIC*. One of the output pins of the serial port is attached to an interrupt-generating input on the MoteNIC. Periodically, the iPAQ raises its output high and records the time according to its CPU clock. The MoteNIC timestamps the resulting interrupts according to its internal clock, and reports the timestamps back to the iPAQ via the serial link. Once the MoteNIC device driver receives this interrupt time back from the MoteNIC, it injects the pair of time values into the time daemon. Each of these points represents a single mapping of the iPAQ's system clock timescale to the MoteNIC's system clock timescale.
- *iPAQ to codec sample clock*. Cheap consumer-grade audio codecs such as the variety typically found in a low-end PC or handheld device tend to have nondeterministic latency bounds when they are asked to start recording or playback. It is important to minimize these effects—delay between the time we ask the codec to start sampling and the time it actually starts contribute directly to errors in the time-of-flight measurement. We have found that this problem can be avoided by running the codec continuously, timestamping each block of audio data as it arrives. Our system includes an “audio server” that continuously records, buffering the most recent 10 seconds of input and making it available to the acoustic ranging process. With the help of a modification to the Linux kernel's codec device driver, the audio server also timestamps each DMA transfer from the codec's chipset as it arrives. It then injects synchronization pairs into the time daemon, each consisting of an audio sample number and corresponding system clock time when the DMA transfer completed.

As these time pairs are fed to the time synchronization daemon, it computes conversion parameters that allow the iPAQ CPU clock to be related to the MoteNIC and codec clocks. 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 or late interrupt on the mote.

These intra-node parameters allow us to complete the synchronization chain: from acoustic mote to MoteNIC (via RBS); MoteNIC to attached iPAQ 1 (via pairs synchronization); iPAQ 1 to iPAQ 2 (via RBS over 802.11), and finally iPAQ 2 to its codec sample clock (via pairs synchronization). The exact synchronization path is computed automatically, transparently to users of the time sync service. This significantly simplifies the localization service, as we will describe in the next section.

## V. LOCALIZATION

The localization subsystem consists of two main components. First, the acoustic ranging component estimates the distances between nodes in the network. Next, a coordinate system is constructed using the range estimates. The following sections describe each of these components in more detail.

### A. Acoustic Ranging

Our acoustic ranging system, described in more detail in [5], uses a wideband pseudonoise sequence to measure the time of flight of sound from one point to another. In the current implementation, the detector is implemented in software, and requires considerable memory and CPU time (far beyond the capabilities of a Mote). However, the high process gain of the detector enables long ranges, accurate phase determination, and excellent noise and multipath immunity. The use of wideband coded signals has two positive effects. The first advantage is that the frequency diversity of the wideband ranging signal enables it to work well in a variety of environments. Because we use audible sound, the wavelengths of sound making up the signal spans from a meter to a centimeter, increasing the resilience of the signal to scattering. The second advantage is that different emitters can select orthogonal codes that can be detected even when they collide. This enables a drastic reduction in the system complexity, as it reduces the need for tight coordination and synchronization.

Because of the limited capabilities of the Mote as an emitter, we emit a position-modulated pulse train by toggling a digital output pin. Passed through the filter of a speaker, each pulse results in a decaying oscillation. The pseudonoise code is represented by variations in the inter-pulse spacing, with an overall rate that is low enough to allow most of the oscillations to decay during each gap. The advantage of this scheme is that, even without exercising control over the dynamics of the speaker, precise timing on the Mote can reproduce the pulse timing with high precision, yielding a high degree of phase accuracy.

The bottom panel of Figure 2 shows how the ranging system works. Timebase conversion metrics between the sender and receiver are maintained by the synchronization service we described in Section IV. To compute the range between two devices, one device sends a message to the other, advertising that it is planning to emit an acoustic signal with a given code at a particular time, referenced in terms of its local timebase. Using the conversion metrics, the receiver can know approximately when to start listening. The audio DSP's are then sampled, and the resulting time series is compared with a locally generated reference signal using a sliding correlator.

In a sliding correlator, the correlation of two signals is computed at different relative phase offsets. Figure 6 shows a graph of correlation as a function of “lag”. By analysing this correlation function, we can estimate the most probable time of arrival of the ranging signal. The maximum value of the correlation function indicates the time of arrival of the strongest component of the signal. Figure 5 shows a portion of the observed signal, aligned with the reference signal at the “best” offset.

Echoes and other environmental distortions result in many successive peaks, among which the earliest peak represents the most direct path. The directional nature of speakers and microphones means that the maximum value of the correlation function often represents a strong reflection; for example, if the speaker is pointed at the ceiling. To resolve this problem, we use an adaptive noise threshold function based on a low-pass filter of the correlation function. We use this low-pass filter to select the earliest “cluster” of peaks, by finding the first region in which the low-pass filter crosses a fixed threshold of 1.5 times the RMS average of the correlation function. Within that region, we select the first peak after the low-pass filter crosses the RMS average.

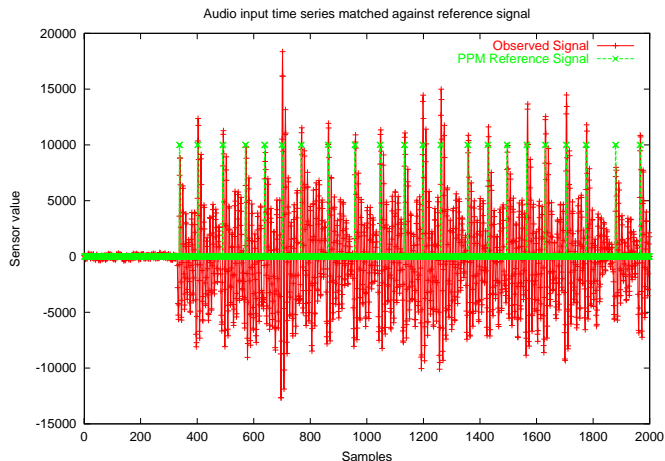


Fig. 5. Pulse Position Modulated reference signal aligned with observed signal, captured in very low noise conditions.

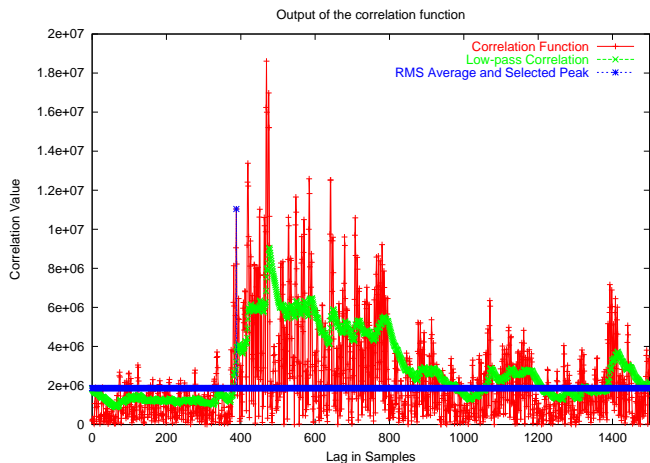


Fig. 6. Correlation function and peak selection algorithm in multipath environment

This algorithm tends to locate the first peaks from the region in which the power level abruptly increases.

Using the timebase conversion metrics and the offset of the earliest peak in the correlation function, the time of flight can be calculated. Given a model for the speed of sound in air, the time of flight can be converted to a distance. Since the speed of sound varies based on temperature and humidity, a value from a temperature sensor may be integrated into the model to get a more accurate distance estimate.

Once distance estimates are collected, multiple trials are combined statistically into a mean and variance, with outlier rejection. However, as we will see below, in general outliers cannot be completely filtered at this layer: long ranges resulting from detection of reflections in obstructed conditions typically exhibit a low variance.

### B. Coordinate system construction

The middle layer in Figure 2 represents the configuration phase of the system. Although we refer to a “configuration phase”, we plan to apply the soft-state design philosophy of treating exceptional cases through the same code paths as typical cases. In such an implementation, there is no distinct state in which configuration is occurring, but rather there is a period of time where a larger number of nodes are joining the system. However, in the current version configuration is done

once at the time of system startup. Future protocol enhancements will support online reconfiguration as the node population changes.

To construct a coordinate system we combine range data from nodes in a given region of space into a single consistent coordinate system. Range data between nodes in the region is collected at a single aggregation point, and an initial coordinate system is accreted around that point. This accretion process is performed iteratively using a “mass and spring” model. The process starts with a fully connected, non-coplanar set of four points, selected to be near (possibly including) the aggregation point. These four points form the kernel of the new coordinate system. Note that there is a mirror-image ambiguity in constructing this system, that cannot be resolved without some kind of 3-D physical geometry, such as a device with three microphones that is solid and does not transmit acoustic energy.

Using the mass-spring system, an initial configuration of nodes is determined. This initial configuration is iteratively improved using non-linear regression to reach a solution that minimizes Gaussian measurement error.

Unfortunately, measurement error is not the only type of error that appears in range data. Several other types of error are present:

- Quantization error on the order of a sample.
- Excess path length caused by clutter, diffraction around objects in the environment, and travelling along surfaces.
- Error caused by the accurate detection of reflections in the case that line-of-sight is obstructed.

Because the conditions that cause excess path length tend to be correlated to the environment, these errors are generally very difficult to remove. Both the spring system and studentized residuals (as suggested in [12]) can be used to detect and eliminate them, although both of these techniques can fail in the presence of multiple ranges exhibiting correlated error.

Figure 7 shows the result of the configuration step with ten receiver devices, compared against ground truth. In the figure, the ‘+’ symbols are ground truth, and the ‘x’ symbols are the receivers. The RMS position error (average per node) in this set of nodes is 11.5cm. The ten receivers are positioned around our lab; none of their positions are known *a priori*. After placing the receivers, we took measurements to establish “ground truth”. Then, each of the ten receivers performed ten acoustic ranging trials, outliers were removed, and the average value of the trials was computed. Thus the ten receivers form a fully connected range graph. In addition to the receivers, several “observation points” are also factored into the minimization. To generate these observations we placed a Mote in various spots in our lab, and took several trials.

In order to relate the coordinate system to our ground truth, we selected an additional four “observation points” and defined their coordinates in terms of our ground truth measurements.

### C. Providing a Location Service

The third layer of our system is the interface that small devices such as motes use to query the system for their location. Unlike those in the configuration step, the algorithms used in this layer are intended to provide faster responses, with an end to supporting the real-time requirements of mobile nodes.

These algorithms consider the locations of the infrastructure nodes to be fixed. The algorithm selects the receivers most relevant to positioning the new node and performs a local optimization to estimate coordinates for the target.

To test our prototype implementation, we performed an experiment to use the infrastructure we just configured. After forming the initial coordinate system, we then placed a mote in several other locations and used the network of receivers to measure its location. The results

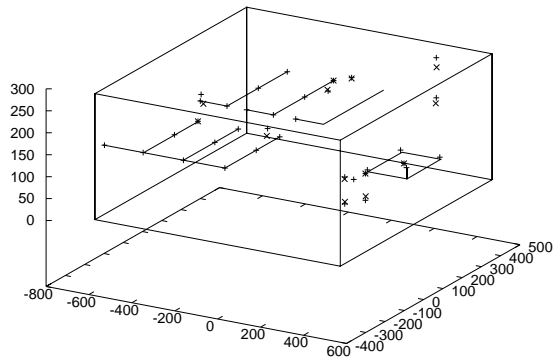


Fig. 7. Positions of receivers after the configuration step. In this diagram only the receivers and four additional points are considered.

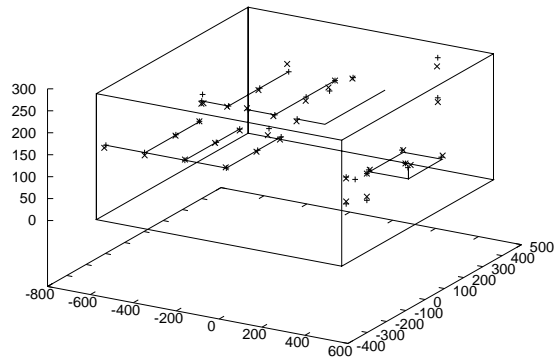


Fig. 8. Positions of receivers and other observed devices. In both diagrams, 'X' indicates computed positions and '+' indicates ground truth. The lines represent cubicle walls and a table in the room.

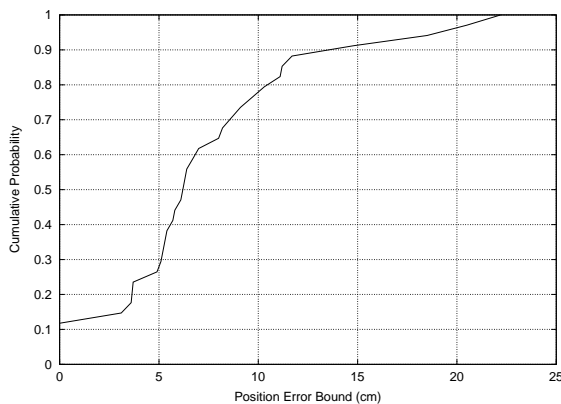


Fig. 9. CDF of position error for the observations in our computed coordinate system. With 80% probability, the position error is less than 10cm.

of these measurements are shown in Figure 8. The RMS position error in this set of nodes is 9.2cm.

In order to evaluate the performance of our localization system, we compared the results produced by our system to our ground truth measurements. For each position that our system estimated, we computed the distance between the computed value and ground truth. A histogram of these values is shown in Figure 9. This histogram plots position error on the X axis, and shows the percentage of measurements exhibiting less than that amount of error on the Y axis.

## VI. CONCLUSIONS

As the size and cost sensors and actuators has fallen, it has become feasible to build distributed sensor nodes capable of being embedded in the environment at high density. In this paper, we present a practical system capable of exploiting that density, achieving  $10\mu\text{sec}$  time synchronization and 10cm spatial localization on a low-cost, low-power, ad-hoc deployable sensor network. This is possible on COTS hardware by making using of novel techniques, including Reference-Broadcast Synchronization and wideband acoustic ranging. We look forward to continuing to develop the platform's capabilities in support of a variety of applications for high-resolution distributed data fusion.

## 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] 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>.
- [4] 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>.
- [5] L. Girod and D. Estrin. Robust range estimation using acoustic and multimodal sensing. In *International Conference on Intelligent Robots and Systems*, October 2001.
- [6] 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.
- [7] 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 Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.
- [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 Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104, Cambridge, MA, USA, November 2000. ACM.
- [9] Chalermek Intanagonwivat, 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 international conference on Mobile computing and networking*, pages 271–278, 1999.
- [11] David L. Mills. Adaptive hybrid clock discipline algorithm for the network time protocol. *IEEE/ACM Transactions on Networking*, 6(5):505–514, October 1998.
- [12] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personal Communications*, 4(5), October 1997.
- [13] 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.