Capturing High-Frequency Phenomena Using a Bandwidth-Limited Sensor Network

Ben Greenstein* Christopher Mar Alex Pesterev Shahin Farshchi
Eddie Kohler Jack Judy Deborah Estrin
University of California, Los Angeles
{ben, cemar, kohler, destrin}@cs.ucla.edu, alex.p@ucla.edu, {jjudy, shahin}@ee.ucla.edu

ABSTRACT

Small-form-factor, low-power wireless sensors—*motes*—are convenient to deploy, but lack the bandwidth to capture and transmit raw high-frequency data, such as human voices or neural signals, in real time. Local filtering can help, but we show that the right filter settings depend on changing ambient conditions and network effects such as congestion, which makes them dynamic and unpredictable. Mote collection systems for high-frequency data must support iteratively-tuned, deployment-specific filter settings as well as fast sampling.

VANGO, our software system for high-frequency data collection, achieves these goals via integrated processing across network tiers. Bandwidth-limited sensor nodes reduce data in network but rely on *microservers*, which have greater computational capabilities and a wider scope of observation, to plan how. VANGO provides a cross-platform library for data transformation, measurement, and classification; a fast and low-jitter data acquisition system for motes; and a mechanism to control mote *and* microserver signal processing. With VANGO we have developed new applications: the first acoustic collection system for motes responsive to changing environmental conditions and user interests, and the first neural spike acquisition application capable of supporting a network of nodes.

Categories and Subject Descriptors:
D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*
General Terms: Measurement, Performance, Design, Experimentation
Keywords: Signal processing frameworks, sensor networks, motes, acoustics, health monitoring

*Contact author.

This material is based in part upon work supported by the National Science Foundation under Grant Nos. 0121778 and 0435497. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

SenSys'06, November 1–3, 2006, Boulder, Colorado, USA.

1 INTRODUCTION

Low power wireless sensor platforms have been designed and deployed to support long-lived, inexpensive, unobtrusive and untethered collection of data from the physical world [13, 15, 25, 26]. Although even the field's first deployed applications sampled the environment at much greater spatial resolutions than previously possible, they have generally been limited to low frequency data (e.g., temperature and humidity) [3, 21, 22]. At the urging of their users, these deployments [11] tend not to process data in-network in ways that could potentially lose information; instead they collect all data samples to a resource-rich server, where a researcher can then search for patterns within a complete data set. At such low sampling rates, the benefits of lossy in-network processing and filtering have not been compelling: there is little data per node to begin with and most systems have not yet hit scaling limits. (Of course, *lossless* compression is valuable even at low sampling rates.)

High-data-rate applications, such as those that involve acoustics, brain waves, seismometers, and imaging, pose new challenges independent of energy considerations. They produce so much data per node that mote radios, with their extreme bandwidth limitations, cannot send that data even one hop in real time. Best-effort collection of such raw data sets will result in significant data loss due to channel contention (presuming a multiple access MAC) and network congestion. Thus, nodes should reduce data before transmission.

If a system must lose data, it is of course better to discard data the user wouldn't want anyway: the system should support *application-specific filtering*. For instance, the users of an acoustic network might not be interested in quiet periods. Unfortunately, filter behavior is hard to predict, and applications will perform poorly when filtering is too aggressive or otherwise poorly calibrated. Sometimes we know a priori the filter parameters that will collect desired information while minimizing transmission energy and bandwidth. More often than not, however, engineering systems need to be tuned while running. Which nodes have interesting data is application- and environment-dependent and time-varying (as our simple experiments confirm). Users interested in the data that sensor networks produce do not know exactly the best way to filter it, because they often haven't seen such spatially dense data before.

Ideally, then, our sensing system should let us experiment with filtering and processing parameters at run time. For instance, when we are calibrating our algorithms, testing hypotheses, or simply searching for interesting patterns within the data, the system could collect raw waveform sam-

ples to a user-accessible device with the processing and storage capability to perform sophisticated signal analysis: a *microserver*. Bringing waveforms back to a microserver lets us easily explore the consequences of different signal processing parameters. When we've discovered a suitable set of parameters and signal processing elements implementable on motes, the system could transfer data processing to the sensor nodes themselves, saving bandwidth and increasing the yield of interesting data. Additional tuning will also occur to compensate for network effects. The calibration process may be repeated when deployment conditions change, or periodically, to maintain confidence in our data.

We have constructed VANGO, a TinyOS-based [18] system for high-rate data collection, around this tuning process. VANGO is quite flexible—it supports several very different applications—while being efficient enough to run on small and very-low-power microcontroller-based sensor nodes. We make four contributions: We have developed the first acoustic collection system for motes that we know of that is flexible to changing environmental conditions and user interests, and the first neural spike acquisition application capable of supporting a network of more than two nodes. We have designed the software system on which these applications run. This system includes a processing library for data measurement, classification, filtering, transformation and compression; a fast and low-jitter data acquisition system for resource-constrained motes; and a mechanism to activate and control mote *and* microserver processing of signals. Finally, we demonstrate through experiments the fidelity tradeoffs in bandwidth limited networks. We show that online calibration of our processing algorithms can dramatically improve the yield of interesting data that is collected; and that in congested networks we should filter our data differently than in unconstrained environments.

2 PLATFORM AND APPLICATIONS

Two target applications motivated our work, each of which collects data at a high rate. The *Auricle* system reports acoustic data, while the *Neuromote* system [4] detects and reports neural spikes generated, for example, by a rat's neurons. Both of these applications are instances of VANGO.

The advantages of a mote including size, cost, deployment flexibility, ability to sleep deeply and wakeup quickly, low energy consumption, and low environmental impact, remain important for these applications. However, mote-like devices, including the relatively advanced TelosB mote we use, cannot continuously transmit at the rates these applications require, and different deployments of either system might require different data compression strategies. Our work shows how a software design can collect high-rate data from motes, even given low-rate radios. This section describes our hardware platform and applications in more depth.

Sensor platform VANGO applications use networks of TelosB motes [25]. TelosB is a state-of-the-art platform for untethered low-power data acquisition. Each TelosB node has 10 KB of RAM, a 250 kbps radio, a 12-channel 12-bit ADC, and a 16-bit MCU running at 4 MHz with no hardware divide or floating point support. For reasons of cost and power consumption, TelosB motes lack application-specific DSPs that could compress raw data in sophisticated ways. Radio bandwidth in TelosB networks can be fundamentally scarce, depending on the number and density of nodes in deployment and on the hardware resident on each node. Traditional techniques for signal compression don't always apply: for instance, GSM could significantly reduce bandwidth for acoustics, but a TelosB cannot run its codec in real time.

Auricle Macroscopic acoustic observations enabled by dense deployments of untethered, unobtrusive sensor nodes could provide scientists with a deeper understanding of wildlife interactions. Proposed acoustic collection projects include monitoring West Coast acorn woodpeckers and marmots [32, 33]. Equally important applications exist in other settings, such as building monitoring and smart spaces.

Auricle collects acoustic data from a network of motes. Each deployed node consists of a mote connected to an amplifier and microphone. Nodes are tasked by, and send processed data back to, a microserver running our software. In general, Auricle brings to audio monitoring the advantages of mote-based sensors, including coverage and scale, low power consumption, low cost, and easy deployment. Auricle is designed to collect raw acoustic waveforms sampled at more than 8 kHz. (The normal range of adult human hearing is approximately 20 Hz to 16 kHz.) Acoustics have heretofore been used by mote-grade platforms primarily for the purpose of target localization; acoustic waveforms are discarded after being measured but before transmission. Interesting data is lost. In this sense, Auricle goes beyond these prior systems [30, 34].

Neuromote Electrophysiological recording is a powerful tool for investigating the mechanisms by which the brain creates and interprets signals. Recordings can help neuroscientists understand the brain function that accompanies emotions, such as fear and aggression, and diseases, such as epilepsy and Parkinson's disease. Neural signals of interest range from an electroencephalogram (EEG), a test to measure electrical activity in the brain (on the order of 10 Hz) to hippocampal fast ripples, high frequency activity (250–500 Hz) of a population of neurons indicative of the onset of epileptic seizure [1]. *Spikes* are waveforms with a period of a couple milliseconds that represent the ion discharge of a single neuron, which normally occur at a rate of 6–10 Hz [36]. To detect neuron spikes, however, a sampling rate of at least 2 kHz is necessary.

Existing wired electrophysiological techniques cannot be used to study freely behaving and interacting test subjects in an enriched natural and social environment, due to the tethering caused by wires and harnesses. Thus, our Neuromote neural sensing application runs on wireless TelosB sensor nodes interfaced with test subjects (such as rats) via implanted depth electrodes and preamplifier circuitry. A Neuromote attachment restricts rat movement and behavior far less than a wired tethering.

Discussion Any system that collects and transmits data at such high rates will clearly use a lot of energy. Energy-limited deployments should sample at high rates only occasionally—with duty cycling, say, or triggered by some other event. We note, however, that Neuromote and similar deployments are not energy-limited as the term is conventionally

understood. Long-term disconnected operation is not important for Neuromote, but form factor and portability requirements require small batteries too weak to power higher-bandwidth radios.

3 CASE STUDY

To understand how VANGO is used to filter data in-network appropriately, we consider a simple Auricle application: collecting the sound of a ringing cell phone.

Ideal environment It is easy to detect, isolate, and collect a ring heard by a single microphone in a low-noise environment. A mote classifies the captured waveform by its amplitude—loud sections likely contain ringing while quiet sections likely do not—and transmits only the loud sections.

Determining the right threshold between quiet and loud is a matter of estimating the ambient noise level and accounting for temporary and unpredictable shifts in that level. In conventional settings, a noise estimator can differentiate noise from signal automatically, so long as the ambient noise level varies slowly and is noticeably below the signal level. In the context of sensor networks, what we consider to be noise—or *uninteresting* data—depends on the user. In an environment with guns firing and cell phones ringing, one user might be interested only in the former, another only in the latter. Determining the threshold between signal and noise therefore must be an *online* process, because the definition of noise changes.

When a cell phone rings in a quiet environment, a mote that uses an amplitude classifier can save energy and reduce radio use by discarding sections of the waveform that aren't loud enough to be interesting. To identify the right threshold between *interesting* and *uninteresting*, a user could run a sample waveform through a classifier running on a microserver, and when reasonably sure that the chosen classifier and parameters will be effective, activate the classifier on the sensor nodes to save energy.

Figure 1 demonstrates this process in VANGO. (See Section 5 for more detail.) The upper sub-figure shows the acoustic waveform of a cell phone ringing as sampled by an Auricle mote and collected to a microserver. The environment has low ambient noise. Analysis on the microserver identifies sufficient classification software (a simple amplitude gate) and parameters (the dashed lines) to detect periods of ringing. This information is forwarded to the mote. The lower sub-figure then shows the system's response to another ring; this time, noise-only periods are pre-filtered out.

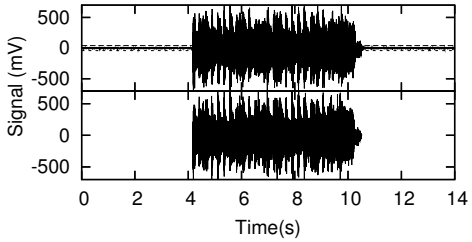

Figure 1—VANGO-collected waveforms with and without amplitude gating. Significant bandwidth and energy are conserved by not transmitting uninteresting portions of the signal.

Environmental interference After some time, the ambient noise level may increase until noise is no longer distinguishable from signal using amplitude alone. More sophisticated processing is needed to isolate the interesting portions of the signal. A waveform sample could be sent to a microserver, which has the resources to, for example, define a convolution filter that removes noise and preserves signal, while being simple enough (of small enough order) to operate in real time on motes.[1] This filter could be applied to the representative waveform in combination with an amplitude classifier to test how well the filter and classifier together isolate the interesting signal. Once the right parameters are found, this processing could be activated on the motes.

Figure 2 shows this process in VANGO. No amplitude gate can selectively filter out noise in a noisy environment (top), so the user uses raw waveforms to develop an appropriate convolution filter (middle). This reveals an amplitude gate level that preferentially selects the signal (dotted lines); installing the filter–gate combination on the motes saves transmissions (bottom).

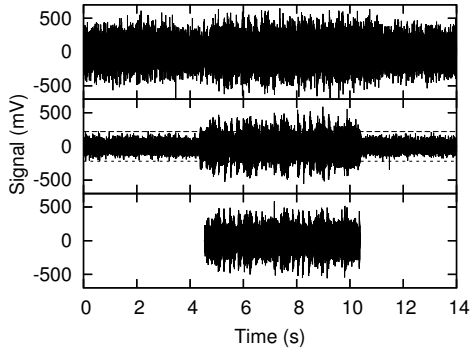

Figure 2—A noisy environment requires an additional convolution filter.

Network interference A microserver receiving cell phone rings from four different motes may notice gaps in all four reconstructed signals due to channel contention. If correlation across sensors is well understood, we could reduce channel contention by disabling sensors that otherwise would produce redundant data. However, correlation is often not well understood, or there may be no redundant sensors. Channel contention can then be reduced only by more aggressive filtering before transmission, discarding *weakly* interesting portions of the signal so as to provide room on the channel to transmit *strongly* interesting portions. Using representative samples taken form each of the motes, a microserver might experiment with filtering levels until data is produced at a rate near the channel's capacity. (The bitrate at which the microserver had been receiving data would be a good estimate of this capacity.) Then filtering might be activated on the motes and fine-tuned to actual network conditions.

Figure 3 shows this process in VANGO. Absent local filtering, multiple motes collecting a signal contend for the channel, leading to random gaps in each collected waveform (a, b). Local filtering can select for data the mote closest to the source (c) at the expense of collecting less information from the more distant node (d).

[1] The order of a filter determines the number of multiplications that must be performed on each sample.

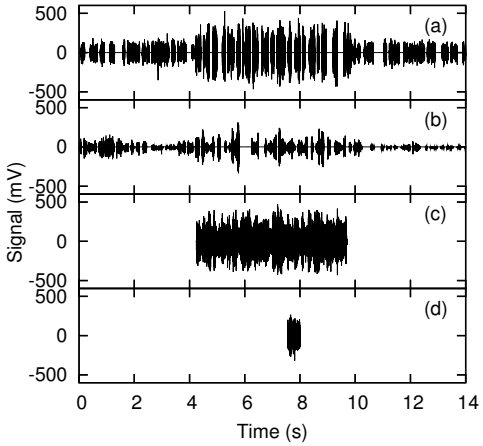

Figure 3—Collection in the presence of congestion.

4 SYSTEM DESIGN

The Auricle and Neuromote applications require support for both interactive experimentation and long-term collection of high-rate data, and sample formats both uncompressed for processing and compressed for transmission. We designed and implemented the VANGO software stack to support these and other high rate aplications. Its basic abstraction, the *sample set*, efficiently provides for sample processing and application-specific format extensions. Each deployed configuration looks like a single filter chain, simplifying control messages (and therefore interactive experimentation) and the construction of new filters. The filter chain abstraction introduces several challenges, including how to process time-series data in discrete windows; how to format data passing throught the system so as to be both flexible and efficient; how to coordinate asynchronous data collection and transmission with synchronous data processing; and how to combine multiple filters.

Figure 4 describes a typical VANGO software configuration. To support tuning and experimentation, the single filter chain spans two platforms. On motes, sensor data is injected into the system by our data acquisition software and pushed synchronously in a single call chain down through the stack of processing filters. At the terminus of this chain sample sets are marshaled into TinyOS packets and transmitted to a microserver via one or more communication hops. A microserver uses a TelosB attached via USB as its network interface; this passes data packets to VANGO's microserver code. From there packets are unmarshaled and passed through a similar processing chain, after which the data is exposed to other applications.

4.1 Data Acquisition

The interrupt load of a sensor network application that interacts with an ADC, radio, and timers will induce significant sampling jitter. Furthermore, the interrupt load produced by an ADC operating at a high rate will overload a system, preventing it from doing much else. To collect high-rate data, therefore, we make use of the DMA controller packaged with the MSP430 MCU on TelosB. We wrote a driver for this DMA and modified existing TinyOS code to use the DMA to coordinate the transfer of samples from ADC conversion registers to sequential words in RAM. As opposed to generating an interrupt after each sample conversion, the DMA generates an interrupt each time it fills a RAM buffer with data. In order to minimize the latency in providing the DMA with a new buffer to fill, and hence to decrease the probability that samples are missed while setting up the next buffer, we prefetch a spare buffer that will be available the instant it is requested.

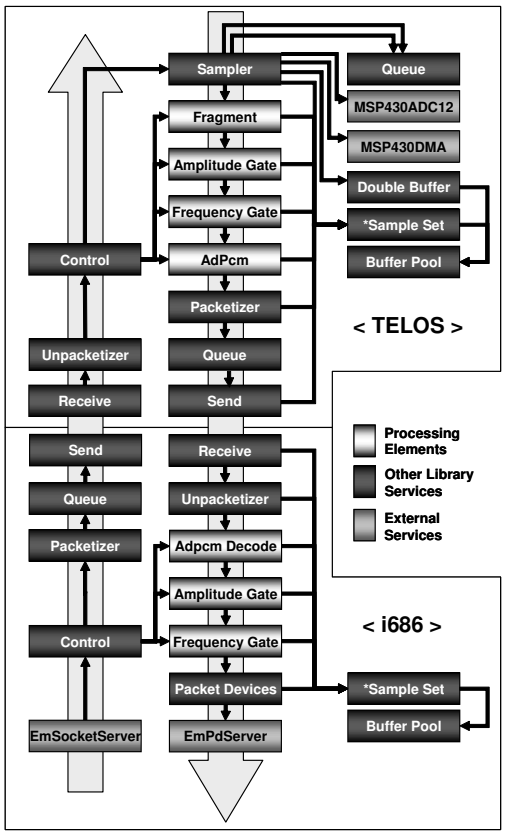

Figure 4—Data and control paths through a VANGO application, including code running on a TelosB mote and on a microserver.

The sampler operates asynchronously, reacting to the DMA's interrupts. We place a queue between the sampler and the rest of the system to isolate this asynchrony. Our queues are designed to work with dynamically allocated buffers; internally, they maintain a circular buffer of pointers to such buffers. To avoid polling by those wishing to dequeue, the queue signals when it becomes non-empty.

We run the data acquisition subsystem with compression software at rates up to 10 kHz. TelosB's ADC is theoretically capable of generating 200 kilosamples per second. If we do not intend to process or transmit data, our acquisition subsystem can sustain rates of up to 115 kHz; insufficient radio bandwidth, however, limits this rate to about 21 kHz (and this presumes that no headers are transmitted and that the channel is perfect and available all the time). In practice, without compression, it is difficult to support above 4 kHz.

For Auricle, the data is generated using a low voltage microphone preamplifier and an omnidirectional condenser microphone.[2] For Neuromote, we AC-couple a pre-recorded

[2]SSM2167 from Analog Devices and WM-61A from Panasonic, respectively, as suggested by a reference design by Moteiv Corporation.

neural signal[3] to a neural preamplifier circuit[4] that we attached to TelosB. The neural signals are acquired differentially from a signal generator as they would be from a live subject.[5]

4.2 Sample Sets

At the core of our system lies the sample set data structure, which contains sensor data and metadata. Sample sets are dynamically allocated from a memory pool and travel through the system much as messages do. Each sample set consists of one metadata buffer and one linked data buffer. The DMA engine described above writes its data directly into a sample set's data buffer. The metadata buffer contains fixed slots for commonly needed information, such as modality, channel, rate, and time, as well as an extensible scratchpad containing type-length-value tuples used by filters. The scratchpad stores intermediate and final processing results. For example, our spike detection algorithm will annotate a sample set with the time, width and height of spikes found in the raw waveform, while the ADPCM codec adds its state to the scratchpad. The scratchpad also provides a convenient way for filters earlier in the processing chain to share data with filters later in the chain. For example, the `Statistics` filter produces a running mean deviation from the mean that is used by the `SpikeDetector` filter. Our Auricle application has 372-byte sample sets, 68 bytes of which are allocated to fixed metadata fields and the scratchpad. The resulting structure offers a tradeoff between flexibility (the scratchpad) and space efficiency (the fixed metadata area).

Library functions marshal and unmarshal sample sets into packet format, facilitating communication with microservers. To make more efficient use of bandwidth, the sample set contains a mask describing which fields to send. For example, the field specifying the sensing modality may be omitted when it is known by the microserver. Sampling and processing components coordinate with the marshalling service to ensure that sample set buffers are allocated to best fit into packets. For example, a system with a 2 to 1 compression algorithm will, at run time, determine that raw data buffers can be twice as large as maximum space available for data in a packet's payload via a chain of per-filter function calls. This chain originates at the data acquisition software, progresses through all filters in the system, and ultimately terminates at the marshaling service. The call path accumulates the aggregate header size required by all filters; on the return path, the packet size less this aggregate header size is scaled in accordance with the data transformations each filter performs.

The sample set is VANGO's universal interface for communicating data between all signal processing components and across platforms. It supplies data to processing components in discrete blocks, which makes it well-suited for batch processing on constrained sensor nodes and for marshaling into radio packets.

[3]Reproduced by a Hewlett Packard 33120A waveform generator.

[4]Analog Devices AD627 instrumentation amplifier with gain set to 200.

[5]The amplified output is referenced to half the battery voltage via a buffered (Texas Instruments OPA234) voltage divider circuit. The DC-referenced output is applied directly to the ADC input of the TelosB mote, while the amplifier ground is shared with the mote ground.

4.3 Filters

VANGO's signal processing takes place on a single linear chain of *filters*, each of which transforms input sample sets into output sample sets. The order of filters in the processing chain can only be changed at compile time and each filter can appear at most once in each platform's part of the chain. Different sensor nodes can have different chains; however, in our applications this hasn't been useful. The linearity and static composition of our processing chains is a limitation of our current software rather than fundamental. However, although linear chaining reduces processing generality, it has not proven to be a limitation for the applications we are targeting. It also has several important advantages, particularly for motes: filters are easy to compose and performance of a linear filter chain is relatively easy to analyze. Since chain reordering at runtime and filter duplication may prove useful as we develop more complicated applications, we are in the process of porting VANGO to the Tenet architecture [9], whose tasking aspects VANGO partially inspired. VANGO filters implemented as Tenet tasks can be duplicated and reordered at runtime.

Not all signal processing algorithms should run as filters on motes, of course. Many of even the simplest signal processing functions are too CPU-hungry to function in real time on a 4 MHz 16-bit processor with no hardware divide or floating point support. For example, consider Fast Fourier Transform (FFT), one of the most basic transformations in signal processing. When optimized for speed, FFT over 512-sample windows of an 8 kHz signal runs on TelosB at one eighth real time (0.064 seconds of data runs in 0.5 seconds.) Furthermore, these algorithms typically use large lookup tables (on the order of 2 kB) for computing sine. We must seek even simpler processing elements that execute quickly, and be willing to trade off some accuracy.

Our filter elements—classifiers, transformation and compression algorithms, and measurements—operate in the time domain directly on sample sets containing raw waveform data, processed data, and measurement results. A data path devoid of signal processing functionality appears like this:

```
Sampler [Filter] -> Packetizer;
```

(The syntax is from SNACK [10].) Filters are added in an application-specific order between the `Sampler`, which acquires sensor data, and the `Packetizer`, which marshals and transmits it. For example, to add ADPCM compression, we write:

```
Sampler [Filter] -> Adpcm -> Packetizer;
```

The mote and microserver halves of Auricle's data processing path are, respectively,

```
Sampler [Filter] -> Stats -> FIR -> AmplitudeGate
        -> FrequencyGate -> Adpcm -> Packetizer;
```

and

```
Unpacketizer [Filter] -> AdpcmDecode -> AmplitudeGate
        -> FrequencyGate -> PacketDevices;
```

(`PacketDevices` exposes data to a user.) The elements in such a wiring are the processing elements that will be built into an application; a runtime control mechanism can dynamically enable, configure, and enable or disable those elements.

We have written filters to analyze and annotate sample sets, to classify sample sets as being worthy of transmission, and to transform sample sets into more parsimonious or revealing formats. The microcontroller utilizations of the filters presented in this paper are summarized in Figure 5, as well as the utilization of our DMA-controlled sampling.

Filter	CPU utilization (%)
ADPCM	43.9
Amplitude Gate	3.3
... with summarization	3.4
Dominant Frequency Gate	4.1
... with summarization	4.1
Statistics	19.9
Format	2.8
Fragment	5.3
SpikeDetector	4.5
... with compression	5.4
FIR (Convolution) order 24	36.3
... order 48	61.1
Sampling (with DMA)	0.3

Figure 5—Worst-case microcontroller utilization of our data processing elements for 304-byte (152-sample) data buffers. The most MCU-intensive filter (FIR with order 48) consumes roughly 61% of available cycles when sampling at 8 kHz.

Measurement filters analyze sample sets, annotating each set with its statistics using the scratchpad. This factors common analysis code out of other filters, which can simply examine the measurement filter's analysis results. Additionally, a filter pipeline might choose to throw out the actual sample data, instead transmitting measured statistics.

We have implemented one measurement filter. `Stats` calculates the running mean, mean deviation from the mean, and standard deviation from the mean for a stream of sample sets and annotates each passing sample set with the current values of these statistics. In its default mode, `Stats` works on groups of 64 samples at a time. It calculates its three statistics for each such group, then adds the three results to three separate exponentially-weighted moving averages (EWMAs). A handful of summary counters are carried over from sample set to sample set, allowing these statistics to be calculated even when sets don't contain multiples of 64 samples. Each sample set is annotated with the three EWMA values after the packet is processed. This informs downstream filters of historical statistics for the sample stream, allowing them to detect unusual deviations. (The configurable EWMA smoothness constant is set to $\alpha = 0.9375$ in our experiments.)

We use `Stats`'s running mean to detect the DC offset of the waveform, a basis for several classifiers that consider signal amplitude and an indicator of biases in the underlying sensing system. This is susceptible, of course, to aliasing effects. Removing aliasing would require extra circuitry and also possibly some additional computationally significant software, such as a convolution filter, depending on the hardware solution. We implemented a heavyweight mote-resident software convolution filter and use it to enhance interesting frequency components of a signal. However, as our sampling rate is generally above the Nyquist rate (eliminating most aliasing effects), the slight improvement in accuracy the convolution filter could bring to a measurement of the mean is not worth the intense load on the microcontroller that convolution incurs.

Classification filters classify each sample set as *interesting* or *uninteresting* by modifying an annotation in the sample set metadata. Each sample set begins as *interesting*, but a classification filter may change the set's annotation to *uninteresting*. Thus, a sample set is marked *interesting* at the end of a filter bank if and only if every intervening classification filter thought it was interesting. The `Packetizer` component only transmits interesting sample sets; uninteresting sets are dropped. Other classifier methodologies are possible, of course. For example, sample sets can start as *uninteresting* and be marked *interesting*; this leads to sets that are interesting if *any* (rather than all) of the classifiers were interested. We have implemented three classifiers, two that are generic and one designed specifically for Neuromote. In each case, the challenge was to implement meaningful classification with minimal computation—for example, to implement a lightweight frequency estimator precise enough to support meaningful classification decisions.

The `AmplitudeGate` filter finds high-amplitude data interesting. It has two parameters: the threshold above the signal mean and the number of samples that must be above this threshold to consider the sample set interesting. The signal mean is read from the `Stats` annotation.

The `FrequencyGate` filter classifies sample sets based on their dominant frequency. It uses perhaps the simplest known time-domain dominant frequency estimator, namely the rate at which the signal amplitude crosses the mean [16]. `FrequencyGate` maintains two exponentially-weighted moving averages of dominant frequency. A sample set is considered interesting if the faster-moving EWMA is within a desired frequency range. Each time the signal transitions to interesting, the slower-moving EWMA is set to the value of the faster-moving EWMA; subsequent sample sets are considered interesting so long as the slow-moving EWMA is still within the desired range. This technique is sensitive enough to avoid missing the beginning of an interesting event in the signal, and provides hysteresis, allowing the gating parameters to be set to a high level while reducing false positives and false negatives. We found empirically by manual experimentation with several acoustic sources similar to those in Section 5 that smoothness parameters of $\alpha = 0.5$ (fast-moving) and 0.96875 (slow-moving) work well; these parameters might need to be retuned for other sources.

Finally, the `SpikeDetector` filter detects single neuron activity in the form of a several-millisecond amplitude spike in the neural signal. The filter's single parameter is the minimum spike height, measured in standard deviations above the mean; any sample above that height indicates a spike. A sample set that contains at least one spike is interesting. The `SpikeDetector` saves information from previous sample sets so as not to miss spikes that occur near or across sample set boundaries.

All of these filters use the statistics generated by `Stats`.

Finally, **transformation and compression filters** alter the input waveform. A convolution filter, for example, transforms an input signal by selectively amplifying and attenuating its frequency components, while compression filters reduce a sample set's resolution, pack its samples more tightly, or compress it using a stateful compression algorithm. The goal is simply to reduce the data that the mote must transmit.

The `FIR` finite impulse response filter transforms an input

signal by direct convolution in the time domain. This technique takes the sum of the dot product of the filter coefficients with a sliding window of the samples. By designing the appropriate filter, we can attenuate arbitrary segments of the frequency band. We designed our filters (generated appropriate filter coefficients) using GNU Octave signal processing functions. The direct convolution technique is well known and simple to implement, although filtering with fast convolution using FFT is more common. Because of hardware limitations, we only use short filters, for which direct convolution is more efficient than FFT in practice.

The `Format` filter alters the precision and alignment of samples within a buffer. `Format` can reduce sampling precision by truncating 12-bit samples to 8 bits each, or reduce waste by packing pairs of 12-bit samples into 3 bytes each. Since `Format`'s output is a valid sample set, annotated appropriately with its precision and alignment, `Format` may appear before or after other filters.

The `Adpcm` filter is an adaptive pulse-code modulation compressor used in Auricle. Adaptive pulse-code modulation is a well-known technique for lossy compression of voice data. When compared to several other compression schemes, including LPC schemes (GSM 6.10) and simple logarithmic encoding (u-law), ADPCM has the best combination of sound quality and compression rate among the few viable for real-time compression on motes. We use a variant of the Intel/DVI ADPCM codec, modified to eliminate multiplication and division operations. Encoding with ADPCM reduces 12-bit ADC samples to 4-bit values. These values are not samples, and cannot be operated on until they are expanded into samples; thus, `Adpcm` must occur last in any filter chain of which it is a part. ADPCM is stateful, so to ensure resiliency to packet loss (and to `Packetizer`'s refusal to transmit uninteresting sample sets), we include the state of the encoder in each packet as a four-byte header extension.

Finally, `SpikeDetector` can be configured to compress as well as classify. When configured in spike-only mode, it filters out baseline noise to produce an abridged version of the signal containing only time-referenced spike waveforms. It also measures each spike's height (amplitude) and width (duration), as these help distinguish the neurons from which it was generated; the sample set is annotated with these parameters, which might obviate most investigators' need for the raw waveform. The resulting data, like that of `Adpcm`, uses a special format, so a `SpikeDetector` in spike-only mode must occur last in the filter chain.

4.4 Tasking and Control

Motes collect, process, and transmit data. Data reception, training of filter parameters, data refinement, data presentation, and the creation and dispatch of control messages are the responsibility of the microservers in our network.

Our applications are comprised of two types of executables: one for the deployed motes to acquire and begin processing the sensor data, and another for the microserver to receive, finish processing and present it. To help integrate these two executables into a single distributed application, we write all our software modules in nesC [6] and compose them into services using SNACK. On microservers, we run nesC code linked to the EmTOS library [7]. From the perspective of a mote, an EmTOS application appears to be another mote; however, on the microserver it appears to be a standard Linux process that may interact with other processes using IPC. Using nesC as the base module description language for our entire system simplifies porting data processing algorithms, control interpretation logic, and link and routing code from motes to microservers and vice versa. In most cases, the port requires no coding changes.

Individual filters may be activated and deactivated and given new parameters at runtime. (A disabled filter passes any sample set it is given to the next filter in the chain without performing any processing.) Control over where and how processing occurs is determined by a user connected to the microserver. Processing on the microserver is invoked using the same tasking syntax as is used to control deployed motes.

Control of the application is exposed via socket so that a human user or controlling application may issue commands from any device with an IP stack. Commands are specified in ASCII and have the following syntax:

```
dest : cmd-name cmd-value [ ; cmd-name cmd-value ]* <CR>
```

For example, to tell all motes to set their amplitude gating threshold to 200 and to enable ADPCM compression, the following suffices:

```
broadcast: gate-threshold 200; adpcm-enable true
```

At the time of this writing, there are about 50 commands defined in our system.

4.5 Communication Patterns

In single-hop scenarios, control messages are delivered directly to intended recipients, either by unicast or broadcast addressing. In multi-hop scenarios, for reliability, control messages are flooded using the Drip [19] dissemination protocol irrespective of the destination address. Relative to the bandwidth consumed by our data traffic, the overhead even of flooding control messages is small.

To avoid expending significant energy buffering sensor data in Flash and to provide acoustics with low latency, data is transmitted by sensor nodes very soon after it is produced. It is collected using the MultiHopLQI routing protocol for TelosB, which is based on Mintroute [39] and supplied as part of TinyOS 1.x [18]. MultiHopLQI forms a collection tree with best-effort transport, as opposed to end-to-end reliability. To support high-rate data transmissions, we modified this code to include a forwarding queue and configured the underlying TinyOS link layer to retransmit at most once. Presuming a clear channel, on TelosB this link layer's CSMA MAC supports a transmission rate close to 80 kbps; on crowded links, this rate can be significantly less (e.g., 50 kbps) as backoff delays induce utilization inefficiencies.

5 Evaluation

This section presents an experimental evaluation of the Auricle and Neuromote applications. We demonstrate that it is actually possible to monitor high-rate traffic over low rate radios: our simple and coarse signal processing filters and classifiers can differentiate interesting data from uninteresting data before it is transmitted, and thus reduce network traffic. In one experiment, traffic was reduced by 78%. Furthermore, we can dramatically improve the effectiveness of

our collection system through runtime configuration of filtering parameters. For instance, by adjusting the gating levels for different collections of motes, we can increase the signal energy we recover by a factor of two.

Our experiments include tests in outdoor and laboratory settings, using single-link and multi-hop communication services, and with uniform filter settings as well as settings that differ from node to node. We evaluate the performance of several combinations of processing components, including `Adpcm`, `Stats`, `AmplitudeGate`, `FrequencyGate`, and `SpikeDetector`, and measure the resulting fidelity tradeoffs in our networks.

5.1 Auricle: Classifier Tuning

We first evaluate how effectively simple local filters can reduce contention in a densely-deployed sensor network. We deploy Auricles sufficiently densely that many nodes can detect source phenomena (i.e., noises), albeit with varying fidelity. We then install `AmplitudeGate`s with varying gating levels; this essentially provides us with noise suppression, silence elimination, and basic event detection. The evaluation measures how much of the source phenomena signal energy is recovered at the sink. Filtering is extremely effective: well-chosen amplitude gate levels can increase this recovered signal energy by factors of 2.5 and more. However, the best amplitude gate is sensitive to user requirements (demonstrating the necessity of on-line tuning) and significantly exceeds the level of ambient noise.

We chose a general metric—recovered signal energy—to show that effective in-network filtering of high-rate data is highly sensitive to environmental factors, particularly RF availability. Of course, in specific application contexts, such as voice recognition or event detection, other metrics would be more telling, like the accuracy of voice reconstruction and percentage of events detected.

Methodology Twelve Auricle motes were situated in a straight line with each pair separated by six feet (Figure 6). A loudspeaker was placed at the end of the line, six feet from the first mote. Each mote was elevated three feet off the ground. Motes transmitted acoustic data directly (one hop) to a microserver sink that was deployed close to the middle of the line. We played two minutes of a recording of former president Jimmy Carter's "Crisis of Confidence" speech, occasionally interrupted by a cell phone ringing; voice and ring volumes were roughly equal. We chose this input data set to demonstrate simple audio feature detection; alternative examples such as gun blasts or birds chirping would also suffice. We sampled at 4 kHz, which is enough to clearly understand a voice, albeit with a noticeable loss of quality. Experiments were run outdoors at night in an open-space environment. Observed sound pressures varied throughout the speech from 78–86 dBC at the closest mote to the speaker to 62–68 dBC at the farthest. The peak-to-peak amplitude of the closest mote's signal measured around 1.22 V, roughly half the range of our ADC. Received signal energy was calculated as $E = \sum s_i^2$, where s_i denotes sample i's AC-coupled value. Only samples collected at the sink were counted. To establish a baseline measurement of the signal energy each node is capable of providing to the sink, we collected recordings from

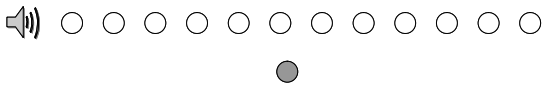

Figure 6—Linear topology used for unicast experiments. White circles represent mote sensor nodes; the grey circle is the microserver sink.

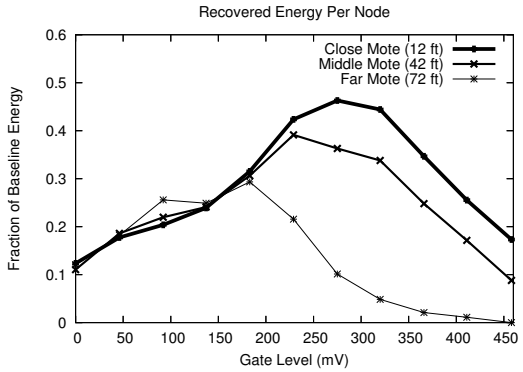

Figure 7—Recovered energy for three nodes of varying distances from an audio source, expressed as the fraction of energy each node could send in an error-free, high-bandwidth environment. Signal energy is maximized by setting the gate threshold just high enough to reduce channel contention. As distance from the audio source increases, peak recovered energy occurs at lower gate levels.

each sensor node individually, with all others silent. Recovered energy readings were normalized using these baseline measurements. The amplitude threshold for `AmplitudeGate` varied from 0 (that is, no gating) to 488 mV above the mean and was uniform across all nodes.[6] In this first set of experiments we seek to quantify how a *globally uniform* adjustment of this threshold impacts performance. Each data point represents an average of 3 trials.

Results We analyze the results from different amplitude gate settings twice, with different goals in mind. First, we aim to recover the maximum *per-node* signal energy; then, we aim to recover the maximum *total* signal energy from the entire network. This is the simplest difference we could envision. Real applications would likely have more complex collection goals, only enhancing the importance of application-specific filter settings we demonstrate via simple experiments.

Figure 7 shows the per-node recovered energy as a function of the amplitude gate's threshold for three nodes: one close to the signal, one far away, and one in the middle. The shape of this graph is dominated by two competing effects. At very low thresholds, little signal is filtered before transmission. This results in channel contention and packet loss; since, in this one-hop scenario, no node gets greater access to the channel than any other, approximately the same fraction of each node's total theoretical signal energy is received. Contention becomes less severe as the threshold increases and silent periods, as well as the lower-amplitude signals at distant nodes, are proactively filtered. Acoustics attenuate rapidly with distance, so nodes farther from the source filter the source more quickly. At very high thresholds, so much data is suppressed at the gate that not only is the channel underutilized, but significant features in the raw signal are removed before transmission.

[6] Since the reference for our 12-bit ADC is 2.5 V, 488 mV corresponds to 800 ADC units above the mean. Calibrating voltages or ADC units to sound pressure levels is a topic for future work.

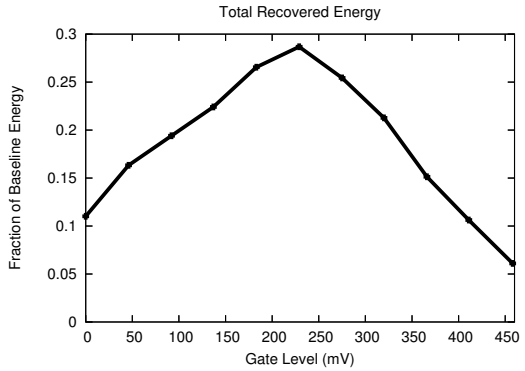

Figure 8—Total energy recovered at the sink from all nodes, expressed as the fraction of the energy all nodes could send in an error-free, high-bandwidth environment. The peak occurs at a gate setting of 230 mV, which does not coincide with the maximum received energy of the mote in the best position to capture the signal (Figure 7). This sharp peak expresses the network-wide point at which congestion effects no longer dominate.

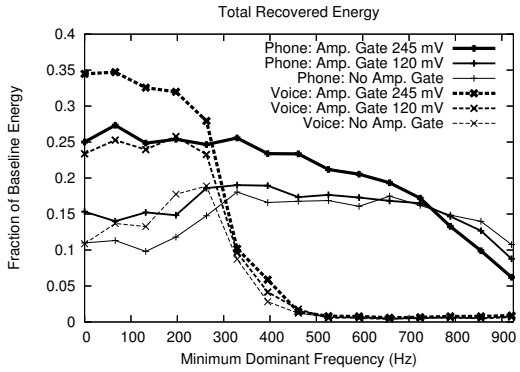

Figure 9—Total signal energy recovered at the sink from all nodes, for interesting ("Phone") and uninteresting ("Voice") signals, as we vary the minimum dominant frequency to pass. Optimizing parameters for a composition of filters cannot be done by optimizing each filter independently. Recovered energy is normalized to the total theoretical energy for the relevant the portion of the signal.

A threshold of 275 mV, the closest node's peak, will maximize the maximum per-node recovered energy. This setting recovers more than 4.5 times the per-node signal energy of the 0 mV threshold (no gating). Although the general trend is intuitive, we expected the maximum signal energy to occur at a threshold just above ambient noise (at about 50 mV). The burstiness of human speech is commonly exploited to optimize voice communication systems; we believed that once our gate eliminated the background noise between words and syllables, the aggregate received energy would trend quickly upwards. Instead, we found that even when gaps between words were filtered, the remaining traffic was still great enough to induce contention-based packet loss and decreased signal yield. Network effects are more pronounced than we had initially expected.

However, users might want a more global metric for yield. Thus, Figure 8 plots the network-wide *total* recovered energy for the experiments of Figure 7. The Y axis shows the total signal energy recovered from the network, as a fraction of the total signal energy theoretically recoverable. In this figure, the peak occurs not at the best per-node threshold of 275 mV, but rather at 230 mV, which filters somewhat less signal from farther nodes. This level recovers approximately 2.5 times the signal energy of a system without gates, and approximately 13% more signal energy than the 275 mV gate.

Without a concrete application it is unclear whether 13% more total signal energy matters, but recovering a factor of 4.5 or 2.5 more signal energy is important for any collection application we can imagine. The gating levels that lead to these improvements depend on network topology (because of contention), signal characteristics (such as attenuation), and environmental characteristics (such as noise). Not all these factors can be known in advance of deployment, and many of them dynamically change, so the ability to dynamically adjust the parameters of in-network filters is crucial.

5.2 Auricle: Filter Composition

We next see whether a combination of simple mote filters can effectively handle a more complex collection task, namely separating cell phone signals from voice signals. High-quality acoustic signal separation is definitely beyond the capabilities of a mote, but can a simpler mechanism drop uninteresting data without dropping too much interesting data? We use `FrequencyGate` filters to discriminate between cell phone rings and speech (rings have a higher dominant frequency). The evaluation measures how well these filters cut back on speech collection, allowing more space for cell phone collection. We find that a combination of frequency and amplitude gates allow motes to improve the signal-to-noise ratio of cell phone signal collection by 13.6 dB, and reduce packets transmitted by 78% over collection without filtering, while not greatly affecting the raw amount of cell phone signal energy collected.

Methodology 33% of our audio recording was silent; during 37% of the recording, Jimmy Carter was speaking without interruption (this includes short periods of silence between words and syllables); and during 30%, the cell phone was ringing over his speech. The cell phone was about as loud as Jimmy Carter's voice. We refer to that entire 30% of the recording as "Phone" signal, and the other 70% as "Voice" signal; note that Phone signal includes some speech as well. While collecting baseline measurements (as before), we noted when the cell phone was ringing, allowing us to determine how much recovered energy is derived from periods when the cell phone is ringing and when it is not. The `FrequencyGate` classifier filters out sample sets whose dominant frequency is below a specified minimum; we vary this minimum from 0 Hz to 920 Hz in steps of approximately 65 Hz. We also include an `AmplitudeGate` classifier, varying its threshold among 0 mV (off), 120 mV, and 245 mV.

Results Figure 9 shows the amount of Phone signal (solid lines) and Voice signal (dotted lines) recovered from the network for various frequency and amplitude gate settings. The amplitude filter is again effective; when the frequency gate is off, Figure 8's best gate level of 245 mV improves both Voice and Phone signal recovery. On top of this, though, the coarse frequency classifier is clearly effective on signals with non-overlapping dominant frequencies: between thresholds of 200 and 400 Hz, unwanted Voice energy sharply decreases while Phone energy stays relatively constant, indicating that the dominant frequency of the former, but not the latter, falls in this range. In general, in order to execute on microcon-

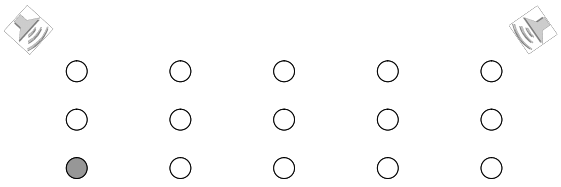

Figure 10—Grid topology used for multi-hop experiments. Diameter is three hops. The grey shaded node represents the data sink.

trollers, VANGO's classifiers tend to be too simple to discriminate much more subtly.

The best setting to remove Voice while retaining Phone signal lies around 500 Hz, depending on the desired compromise between false negative and false positive readings. With an amplitude gating level of 245 mV and a minimum dominant frequency level of approximately 525 Hz, almost all Voice periods are suppressed: the ratio of Phone to Voice energy is approximately 16.6:1, giving a signal-to-noise ratio (SNR) of 12.2 dB. By comparison, when the dominant frequency filter is effectively off, the energy ratio is 1:1.4, giving a SNR of -1.39 dB. Therefore, optimizing the dominant frequency filter parameters results in a 13.6 dB improvement in SNR. More concretely, this corresponds to a reduction in packets transmitted by 78%.

Even with a perfectly tuned dominant frequency classifier, an amplitude classifier can help increase the yield of interesting data. Even after all Voice data has been suppressed, if several nodes detect the same Ring waveform, their transmissions may contend with one another for the radio channel, resulting in data loss. Since motes have limited RAM (10 kB in our deployments) and cannot queue their data for long (1 second when sampling at 4 kHz), correlated detections can lead to channel contention even when acoustic events are rare and short-lived. Hence, proactively filtering *weakly* interesting signals so that *strongly* interesting ones have a greater chance of being received will improve the yield of signal energy. Figure 9 shows this in practice: after removal of all Voice signal, amplitude gating improves Phone signal yield by 30%. Of course, as we proactively filter more and more of our signal using the dominant frequency classifier, the maximum channel utilization during any period will eventually drop below 100%; at that point (roughly 725 Hz) amplitude gating is no longer useful.

5.3 Auricle: Multi-Hop Collection

We now investigate whether our filters make it possible to collect acoustics signals over multi-hop networks, and turn our focus from a dense single-hop deployment to a sparser multi-hop deployment. Can a simple classifier reduce unwanted traffic enough to significantly increase a multi-hop network's yield of interesting data? Does the distance of a sensor node from a collection gateway affect how this classifier should be tuned? We use the `AmplitudeGate` with varying gating levels to discriminate *strongly* interesting speech and cell phone rings from all other noise sources. We find that although there is a noticeable performance decrease with multi-hop networking, the integrated system as a whole operates well. Even in the presence of significant unwanted traffic generators, filter parameters can be tuned to recover 5 times the interesting signal energy generated by distant nodes over what can be recovered without filtering.

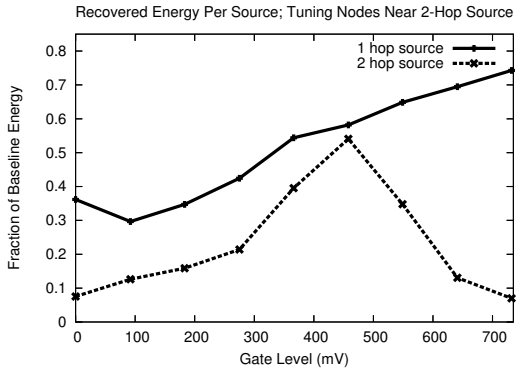

Figure 11—By fixing the gating level at an approximately optimal level of 275 mV for half of the network around the nearby node (1-hop source), and adjusting the gating level for the other half of the network, we achieve approximately 55% of the theoretical energy.

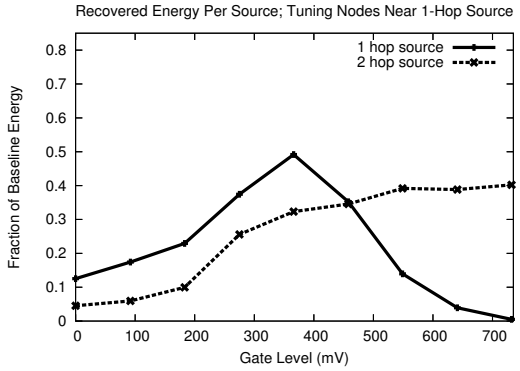

Figure 12—Fixing the gating level at 275 mV for half of the network around the faraway node (2-hop source), and adjusting the gating level for the other half of the network, we achieve at most approximately 40% of the theoretical energy. The difference between performance for the nearby and faraway sources is packet drops on bottlenecked forwarding nodes.

Methodology Fourteen TelosB motes were deployed in a 3-by-5 grid, with nodes separated by 25 and 8 feet along the length and width respectively. A sink was deployed at one corner of the network, with sources six feet from the nodes at two other corners; see Figure 10. Nodes close to one source had one-hop connectivity to the base station, while nodes close to the other had two-hop connectivity. Since the likelihood of losing a packet to a forwarding queue overflow is independent of the packet's source, delivery probabilities along longer paths (3 hops and more) is very low when many motes are generating traffic. We don't present experiments with these longer paths as lack of flow-level fairness dominates system performance, an issue more for transport protocol design. We placed the nodes directly on the ground to increase RF attenuation, simplifying deployment mechanics.[7] For tree-based collection, Auricle uses MultihopLQI [39] augmented to properly queue high-rate traffic. To repeat experiments, we froze the routes once they stabilized.

We ran two sets of experiments for amplitude gate settings. The nodes were divided into two groups, depending on which source (the 1-hop source or the 2-hop source) was physically nearest. In one set of experiments, we set the 1-

[7]When nodes are placed a meter above the ground, the nominal radio range of the TelosB is between 300 and 600 feet depending on environmental conditions. Placing the nodes directly on the ground decreases this range to about 50 feet.

hop source group's gate to 275 mV and varied the gate for the 2-hop source group; in the other set, we did the reverse, fixing the 2-hop source group's gate and varying that of the 1-hop source group. This procedure was designed to evaluate the impact of hop distance on optimum gating level. As before, we compare against baselines measured once per node in the absence of contention; the network-wide baseline energy is approximately equal for each of the two acoustic sources. Each point represents the best result of five trials.

Results Figures 11 and 12 show the results. The maximum total recovered energy, 55% of the theoretical energy, is obtained when the 1-hop and 2-hop source groups have amplitude gates 275 and 450 mV, respectively; this point is visible as the 2-hop source's peak in Figure 11. The gating level for the 2-hop source group is higher than that for the 1-hop source group probably because network contention impacts multi-hop transmission proportionately more than single-hop transmission. This is further visible in the differences between Figure 11 and 12 at high gating levels: even when all of the 1-hop source's signal is filtered out, the 2-hop source with gating level 275 mV (right-hand side of Figure 12) achieves less recovered energy than it does with a higher gating level in Figure 11. Aggressive, topology-dependent filtering can thus improve the recovered energy from a network, and we have shown that filter tuning can lead to a system that can successfully collect high-rate signals over low-rate radios, even over multiple hops.

5.4 Neuromote

The Neuromote application collects neural signals in real time. In this section we study how effectively VANGO can reduce network traffic while still capturing interesting neural spike information. We expect that at high sampling rates, our network does not have the bandwidth to collect complete neuron waveforms from several nodes in real time; however, low sampling rates will result in poor spike detection and characterization. Can we use the `SpikeDetector` filter to accurately detect and collect spikes from several sources, and can we help a scientist to trust the data? We apply the `SpikeDetector` and adjust the sampling rate. The evaluation measures the key parameters of interest to a scientist: spike heights and widths, as well as the percentage of spikes that are detected and recovered. We find that with the right parameters, a Neuromote network can accurately collect neural data from as many as eight concurrently monitored test subjects. Furthermore, we find that the process of determining the right filtering parameters—interactive and iterative refinement—also instills a high degree of confidence in the data the network produces.

Methodology Our experimental setup consists of a waveform generator programmed to output pre-recorded neural signals, a neural preamplifier circuit, and eight TelosB motes. The data programmed into the waveform generator was originally acquired in vivo from freely moving rats.[8] Data were recorded wide band (0.1 Hz to 5 kHz) and sampled at 10 kHz with 12-bit precision.[9] The data set corresponds to one second of neural activity over which there are seven spikes, each with an amplitude of 1.85 V (at the motes' ADC inputs) and a peak-trough duration of approximately 1.34 ms.

We performed two sets of tests. One test measured the percentage of spikes recovered for different sampling frequencies and numbers of motes communicating in the network. This test was performed in two modes of operation. In the raw mode, the entire sampled signal was transmitted by each node. In the spike-only mode, motes transmit only the portions of the waveform that contain spikes. The second test measured the extent of spike parameter variation resulting from different sampling rates on one to eight motes. The spike parameters of interest are (a) spike height, which is the voltage of the acquired signal peak, and (b) spike width, which is the time difference between the spike's peak voltage and minimum voltage. Spikes that were lost due to packet loss are not accounted for in this test. The sample data used contained spikes from a single cell; therefore, each spike was originally equal in amplitude.[10]

Results Figure 13 describes the percentage of recovered spikes as a function of the number of nodes in the network at sampling rates of 2 kHz and 8 kHz. When sending the complete raw data set, we find that the network has the bandwidth to support one or two motes sampling at 2 kHz, returning nearly 100% of the data pertaining to spikes. However, even at a 2 kHz sampling rate, with three transmitting motes the spike delivery rate drops off significantly (to under 60%) and decays to 20% when eight motes are active in the network. In terms of the total number of spikes returned by the network, when sending raw data we find that the maximum spike yield occurs with two motes.

In contrast, when spike information is concatenated and the noise between spikes is removed before transmission, the network performs significantly better. At 2 kHz our yield is above 90%, even for eight motes. At 8 kHz the network suffers a bit from contention and the spike recovery rate decays linearly from 100% with two motes to under 70% with eight motes. In terms of the total number of spikes received, we see that the maximum total spikes recovered (with 8 motes at 8 kHz) is roughly two and a half times greater than the maximum for our raw data experiments (with 2 motes at 2 kHz).

Figure 14 describes the effects of sampling rates on networks of one, two, and eight motes. Irrespective of whether raw or abridged waveforms are sent, at below 2 kHz we witness considerable spike loss due to undersampling. This figure again shows that the network can support two nodes' worth of raw data at up to a sampling rate of 2 kHz. This suggests that when sending the complete waveform the only operating point where nearly all spikes are detected is at 2 kHz with exactly two motes.

Figure 14 also shows that network congestion penalizes requests for raw waveforms. This penalty becomes more pro-

[8] Using five four-channel MOSFET input operational amplifiers mounted in the cable connector to remove movement artifacts.

[9] The spikes were isolated from the local field potentials by applying a high-pass filter with an f-3dB frequency of 600 Hz. The output signals from the waveform generator are applied to the neural preamplifier circuit, which amplifies and DC references the signals, which are then applied directly to the ADC inputs of all eight TelosB motes.

[10] However, the the finite resolution and sampling rate of the original neural signal acquisition apparatus results in a deviation of 108 mV in the data set that has been programmed into the waveform generator.

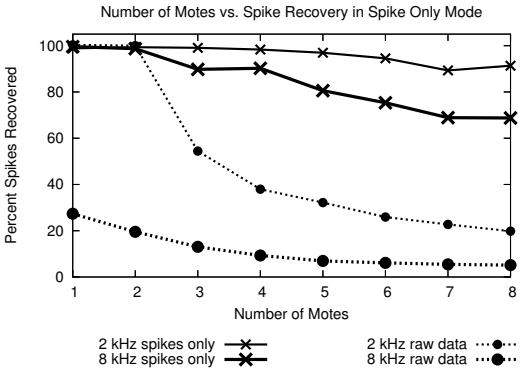

Figure 13—Percentage of recovered spikes as a function of the number of nodes in the network. In the continuous signal transmission mode (dotted lines), 100% of spikes are only recovered with up to 2 motes, both of which must be sampling at 2 kHz. However, when only the waveform containing spike information is transmitted (solid lines), our spike recovery rate is near 100% for up to six motes sampling at 2 kHz and always above 65% when sampling at 8 kHz

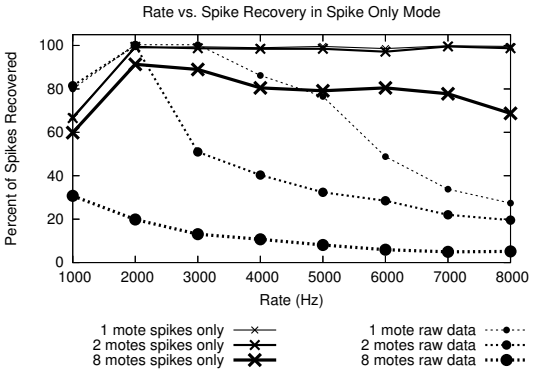

Figure 14—Percentage of recovered spikes as a function of sampling rate and node count in both continuous signal (dotted lines) and isolated spike waveform (solid lines) transmission modes. In mote modes, spike loss is observed below 2 kHz due to under-sampling. At 2 kHz and above, due to channel contention the spike recovery rate drops with rate, but much less significantly with abridged data. Packet loss is worse for larger networks.

nounced both as the number of motes and as the sampling rate is increased. However, in the isolated spike waveform transmission mode for 1 and 2 nodes the recovery rate is near 100% irrespective of sampling rate. We receive nearly 80% of the spikes when running eight motes in this mode.

Figures 15 and 16 show how the key scientific parameters recovered from raw waveforms—namely spike widths and heights, respectively—differ as the sampling rate changes. As expected, the greatest amount of variation can be observed when the signal is being undersampled (1 kHz). The parameter variation drops off with increasing sampling frequency, leveling off between 2 and 3 kHz for both parameters. At 3 kHz the variation of the spike heights and widths approach those of the original pre-recorded data set.

Thus, with the right settings—a sampling rate of 2 kHz and transmission of the abridged neural waveforms only—Neuromote can capture the neural spikes from eight subjects concurrently and accurately.

Neuromote, like Auricle, shows the tension between filter lossiness and network contention. The right filtering point depends on many factors, so a priori knowledge is often insufficient to determine the correct filtering parameters. Our results shows that system performance follows general trends

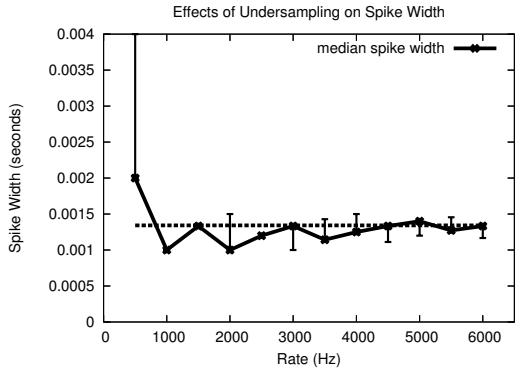

Figure 15—Median spike width as a function of sampling rate. The error bars indicate the 1st and 3rd quartiles. The dotted line at 0.0013 seconds represents the median width for all spikes in the input. At frequencies below 2 kHz, the variation in the recovered spike widths is high due to undersampling. As the sampling rate is increased, the reported median spike width converges to the correct value. Recovering the median spike width with increased accuracy enables investigators to classify the cell from which the spike pattern originated with greater precision.

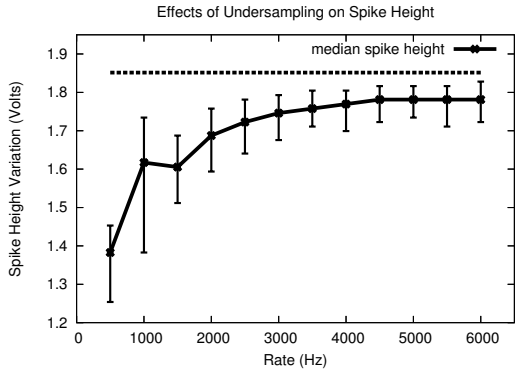

Figure 16—Median spike height as a function of sampling rate. The error bars indicate the 1st and 3rd quartiles. The dotted line at 1.85V represents the actual median height for all spikes in the input. At frequencies increase, the accuracy of the reported spike heights increases substantially. Accurate reporting of spike height enables investigators to classify the cell from which the spike pattern originated with greater precision.

with respect to filter settings. Thus, a few iterations of online tuning will usually yield a significant improvement.

In addition, the collection and tuning process helps scientists to trust the system. If nodes transmits filtered data, how can a scientist know if the signal is being accurately represented? How does the scientist know that the sensor is connected correctly to a rodent's brain (in particular, that the attachment isn't too close to neighboring neurons), that the sampling rate is sufficient for characterization of the signal, and that the network has enough bandwidth to transmit information from several rodents concurrently? With Neuromote, scientists may collect complete neural waveforms from one or two nodes and adjust the sampling rate until the signal is verified to be accurate. The network may then be instructed to eliminate periods of noise between neural spikes to save bandwidth.

6 Related Work

6.1 High-Rate Sensor Network Applications

Several other systems sample at high rates, but usually target very specific applications or system services (localization in particular), provide few configuration knobs, and collect

summary interpretations of the waveform, not the sampled waveform itself.

Acoustics Acoustic sensor networks are typically factory-tuned to detect specific events. For example, several indoor localization systems record ultrasonic pulses for the sole purpose of noting their times of arrival. Cricket [27], Active-Bat [35], and AHLoS [29] use these times to estimate the distance to a sender. While VANGO does not generate acoustic and ultrasonic waveforms, it can detect and timestamp them. More work would be needed to determine if VANGO's accuracy is sufficient for ranging. Likewise, the counter-sniper system [30] uses mote networks and significant complementary signal processing hardware (an FPGA on a custom-designed sensor board) to detect gunshots and localize their sources. While VANGO's spike detection software could be applied to detect gunshots, the 1 MHz required sampling rate to perform fine-grained acoustic localization is beyond the capabilities of the mote hardware VANGO uses. VigilNet [12], a sensor network surveillance system, uses dual axis magnetometers, microphones, accelerometers, and photo sensors to detect similarly application-specific events and then send notifications. It can trade detection sensitivity for longevity, but its design is otherwise rigid. VigilNet's vehicle detection logic, wakeup message format, logic to define the group of nodes that are to be recipients of such messages, protocol for duty-cycled rendezvous, and in-network cooperating aggregation of tracking messages are all application-specific and complicated; reusing the code in new applications will be difficult. This contrasts the work of VANGO, which provides software that is more generic and reusable. Finally, ENS-Box [8] is a flexible platform for prototyping rapidly deployable acoustic sensing systems that do significant in-network processing. Unlike VanGo, it is built atop Linux and is designed with greater computational capabilities (ARM hardware) in mind.

Biological signals The CodeBlue [20] project has developed mote-based sensors with biological interface circuits, some similar to those used for VANGO's neural monitoring application. These nodes have been used to acquire and wirelessly transmit biological signals, including pulse oximetry and electrocardiogram (EKG) data. CodeBlue, however, was not designed to acquire and filter data at the rates necessary, for example, to obtain and transmit neural spike activity. The EKG waveform, for example, is typically sampled at 120 Hz.

Other high-rate phenomena The Cyclops imaging system [28] produces low-frequency data but at a high resolution. Data reduction before transmission is thus often key to promoting network longevity and meeting available bandwidth. Cyclops is capable of performing a range of data acquisition and manipulation options, but it has not yet been successful in exposing this functionality in a manner which can be composed at runtime to meet new application tasks; we hope the data structures and filter organization of VANGO will be helpful in this regard.

Accelerometers and cheaper seismoacoustic sensors have been used in structural monitoring and the measurement of seismic activity. Werner-Allen et al. [37, 38] deployed two mote-based seismic arrays on volcanoes in Ecuador. Their nodes sample at roughly 100 Hz. To reduce data in-network, they use a detector similar to VANGO's spike detector. Kim et al. [17] monitored ambient vibration of the Golden Gate Bridge using Mica2 motes and custom accelerometer boards. Using carefully written embedded code, they were able to achieve a sampling rate of 200 Hz with relatively low jitter—much less of a problem with the newer TelosB mote and DMA-based data acquisition software of VANGO. Likewise, Wisden [41], a system for reliably transporting structural vibration data from a collection of sensors to a base station, incorporates Mica2 motes and a vibration card (accelerometer), samples at a high rate (100Hz), and delivers data reliably over multiple hops to a base station where it can be visualized.

6.2 Signal Processing Frameworks

Many other signal processing compositional frameworks exist. These systems typically offer extensive high-level libraries and streamlined user interfaces. Their sophisticated user-level support is complementary to the work of VANGO. While they are sometimes capable of producing code for embedded platforms, they are not designed explicitly for operation on distributed microcontroller-based wireless sensing systems. Labview is a platform with a visual dataflow language used for data acquisition, industrial automation, and instrument control [14]. Labview applications depend on a runtime engine and libraries designed for PCs. Simulink [23] and Ptolomy II [2] are visual component composition systems for modeling control and signal processing applications. When Simulink is used with the real-time processing toolbox, it can generate high performance code that is suitable for constrained embedded systems. Similarly, Ptolemy applications can be compiled for an embedded target. StreamIt [31] is a language and compiler for generating real-time streaming systems on embedded platforms. It supports general purpose uniprocessors and the MIT Raw machine.

6.3 Internet and Cellular Transcoding

The problem of reducing high-rate data to alleviate network congestion has been studied extensively in the context of Internet and cellular transcoding [5, 24, 40]. VANGO shares one goal in particular with these projects, maximizing application performance given bandwidth constraints. Unlike the previous work, however, a sensor network application competes with itself for bandwidth. This property alone can lead to finer control over how data is collected and filtered, because a sensor network application can make network-wide decisions about how to allocate and use bandwidth.

7 CONCLUSION

We have designed a heterogeneous software system capable of high-rate data acquisition and single- and multiple-hop wireless transmission. Not only does it efficiently process, classify, measure, compress, and transform raw sensor data, but it also provides cross-tier mechanisms to help users calibrate the system while it is running and thus improve its performance. We presented high-rate collection applications in two domains, acoustics and neurophysiological monitoring. In both applications, simple filters can impact network

performance greatly, particularly in bandwidth-limited environments. As we have shown with both applications, selective and informed filtering before transmission yields more of the data that interests a user. A system designed around the needs of both high-rate sampling and flexible runtime filter tuning makes the benefits of mote data collection accessible even to high-data-rate applications.

ACKNOWLEDGMENTS

We would like to thank our reviewers and our shepherd, Sam Madden, for valuable feedback; Mohammad Rahimi for hardware support; and the TinyOS and Emstar communities for software contributions.

REFERENCES

[1] A. Bragin, J. Engel, C. L. Wilson, I. Fried, and G. W. Mathern. Hippocampal and entorhinal cortex high-frequency oscillations (100–500 Hz) in human epileptic brain and kainia acid-treated rates with chronic seizures. *Epilepsia*, 40:127–137, February 1999.

[2] J. Buck, S. Ha, E. A. Lee, and D. G. Messerschmitt. *Ptolemy: A framework for simulating and prototyping heterogeneous systems*, pages 527–543. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

[3] A. Cerpa, J. Elson et al. Habitat monitoring: Application driver for wireless communications technology. In *Proc. 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean*, April 2001.

[4] S. Farshchi, P. H. Nuyujukian, A. Pesterev, I. Mody, and J. W. Judy. A TinyOS-based wireless neural sensing archiving and hosting system. In *Proc. 2nd International IEEE EMBS Conference on Neural Engineering*, March 2005.

[5] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proc. 16th ACM Symposium on Operating Systems Principles*, pages 78–91, Oct. 1997.

[6] D. Gay, P. Levis et al. The nesC language: A holistic approach to networked embedded systems. In *Proc. ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation (PLDI)*, pages 1–11, June 2003.

[7] L. Girod, T. Stathopoulos et al. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proc. 2nd ACM SenSys Conference*, pages 201–213, Nov. 2004.

[8] L. Girod, M. Lukac, V. Trifa, and D. Estrin. The design and implementation of a self-calibrating distributed acoustic sensing platform. In *Proc. 4th ACM SenSys Conference*, Nov. 2006.

[9] O. Gnawali, B. Greenstein et al. The Tenet architecture for tiered sensor networks. In *Proc. 4th ACM SenSys Conference*, Nov. 2006.

[10] B. Greenstein, E. Kohler, and D. Estrin. A sensor network application construction kit (SNACK). In *Proc. 2nd ACM SenSys Conference*, pages 69–80, Nov. 2004.

[11] R. Guy, B. Greenstein et al. Experiences with the Extensible Sensing System ESS. Technical Report 61, CENS, UCLA, Mar. 29 2006.

[12] T. He, S. Krishnamurthy et al. VigilNet: An integrated sensor network system for energy-efficient surveillance. In *Proc. 3rd ACM SenSys Conference*, Nov. 2005.

[13] C. T. Inc. `http://www.xbow.com/`.

[14] N. Instruments. Labview – the sofware that powers virtual instrumentation. `http://www.ni.com/labview/`.

[15] J. Kahn, R. Katz, and K. Pister. Next century challenges: Mobile networking for Smart Dust. In *Proc. 5th Annual International Conference on Mobile Computing and Networking (MobiCom '99)*, pages 271–278, Aug. 1999.

[16] B. Kedem. Spectral analysis and discrimination by zero-crossings. *P-IEEE*, 74:1477–1493, 1986.

[17] S. Kim. Wireless sensor networks for structural health monitoring. Master's thesis, University of California, Berkeley, 2005.

[18] P. Levis, S. Madden et al. The emergence of networking abstractions and techniques in TinyOS. In *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI '04)*, pages 1–14, Mar. 2004.

[19] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI '04)*, Mar. 2004.

[20] K. Lorincz, D. J. Malan et al. Sensor networks for emergency response: Challenges and opportunities. *IEEE Pervasive Computing*, October–December 2004.

[21] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Dec. 2002.

[22] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, Sept. 2002.

[23] MathWorks. Simulink – simulation and model-based design. `http://www.mathworks.com/products/simulink/`.

[24] S. McCanne, V. Jacobson, and M. Vetterli. Receiver-driven layered multicast. In *Proc. ACM SIGCOMM '96 Conference*, pages 117–130, Aug. 1996.

[25] J. Polastre, C. Sharp, and R. Szewczyk. URL `http://www.moteiv.com`. Moteiv website.

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

[27] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proc. 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, Aug. 2000.

[28] M. Rahimi, D. Estrin, R. Baer, H. Uyeno, and J. Warrior. Cyclops, image sensing and interpretation in wireless networks. In *Proc. 2nd ACM SenSys Conference*, pages 311–311, Nov. 2004.

[29] A. Savvides, C.-C. Han, and M. B. Srivastava. Dynamic fine-grained localization in ad-hoc networks of sensors. In *Proc. 7th Annual International Conference on Mobile Computing and Networking (MobiCom 2001)*, pages 166–179, July 2001.

[30] G. Simon, M. Maroti et al. Sensor network-based countersniper system. In *Proc. 2nd ACM SenSys Conference*, pages 1–12, Nov. 2004.

[31] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A language for streaming applications. In *Proc. 2002 International Conference on Compiler Construction*, pages 179–196, April 2002.

[32] V. M. Trifa. *A framework for bird songs detection, recognition and localization using acoustic sensor networks*. PhD thesis, Ecole Polytechnique Federale de Laussanne and University of California, Los Angeles, 2006.

[33] H. Wang, C. Chen et al. Acoustic sensor networks for woodpecker localization. In *SPIE Conference on Advanced Signal Processing Algorithms, Architectures and Implementations*, Aug. 2005.

[34] Q. Wang, W. Chen, R. Zheng, K. Lee, and L. Sha. Acoustic target tracking using tiny wireless sensor devices. In *Proc. 2nd International Symposium on Information Processing in Sensor Networks (IPSN '03)*, pages 642–657, Apr. 2003.

[35] A. Ward, A. Jones, and A. Hopper. A new location technique for the active office. *IEEE Personnel Communications*, 4(5):42–47, October 1997.

[36] P. T. Watkins, G. Santhanam, K. V. Shenoy, and R. R. Harrison. Validation of adaptive threshold spike detector for neural recording. In *Proc. 26th International Conference of the IEEE EMBS*, pages 4079–4083, 2004.

[37] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Proc. 2nd European Workshop on Wireless Sensor Networks*, January–February 2005.

[38] G. Werner-Allen, K. Lorincz et al. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, March–April 2006. Special Issue on Data-Driven Applications in Sensor Networks.

[39] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. 1st ACM SenSys Conference*, Nov. 2003.

[40] D. Wu, A. Swan, and L. A. Rowe. Internet MBone broadcast management system. In *Proc. ACM SIGMultimedia Conference on Multimedia Computing and Networking*, 1999.

[41] N. Xu, S. Rangwala et al. A wireless sensor network for structural monitoring. In *Proc. 2nd ACM SenSys Conference*, Nov. 2004.