

UCLA

Technical Reports

Title

Collecting High-Rate Data Over Low-Rate Sensor Network Radios

Permalink

<https://escholarship.org/uc/item/2nj1r0x8>

Authors

Ben Greenstein
Alex Pesterev
Christopher Mar
et al.

Publication Date

2007

Collecting High-Rate Data Over Low-Rate Sensor Network Radios

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

ABSTRACT

Embedded systems can already capture data produced at high rates, and embedded CPU and sensor performance are still rapidly improving. Radio technology, however, can not keep pace, and will not in the future due to known physical limits of shared communication channels. This leads to a fundamental gap between the data a sensor network node can collect and the data it can transmit back for analysis.

VanGo, our software system for data collection, uses flexible transcoding to narrow this gap. To make effective use of channel bandwidth, data reduction software must run on sensor nodes. However, to calibrate *how* data reduction software should run, that same software should be capable of running on the back end on real data received from the network. In VanGo, users decide where data processing occurs.

To show that transcoding helps, we evaluate two radically different applications: acoustic collection and the measurement of neural activity. Among our findings is that in bandwidth-limited environments, proactive filtering of some of our signal can result in collecting three times the signal energy than we could by removing silent periods alone.

1 INTRODUCTION

Sensor networks are envisioned to provide long-lived, inexpensive, unobtrusive and untethered collection of sensor data (e.g., near bird nests or on rodents' backs) [22]. Low-power platforms with limited computational and communication facilities have been developed to this end [12, 14, 21].

Although even the first deployed applications sampled the environment at much greater spatial resolutions than previously possible, they sampled at very low rates (on the order of seconds or minutes between samples) [2, 18, 19]. While these applications proved that it was possible to use very simple devices to collect and transfer data over multi-hop wireless networks, they did not test the responsiveness and robustness of these networks in the face of intense traffic bursts and congestion. Moreover, for the sake of simplicity and rapid deployment and at the urging of the their users, these deployments [4] tend not to process data in-network, instead opting to collect all data samples to a resource-rich

back-end, where a researcher may at her leisure discover interesting patterns within a complete data set. At such low sampling rates, the cost benefits of in-network processing and filtering that reduce datasets *in situ* to save energy do not come into play because there is little data per node to begin with and systems have not yet grown to be very large scale.

High data-rate applications such as those that involve acoustics, brain waves, seismometers, and imaging pose new challenges. They do produce lots of data per node, so much so that given the bandwidth limitations of extremely low power radios, it is sometimes impossible to collect in raw form all of the data produced; best effort collection of raw datasets will result in significant data loss to channel contention and network congestion.

In these applications it often makes sense for nodes to process data before transmission, because the concise summarizations that may be produced from raw waveforms may yield more information than a raw waveform made incomplete by excessive network congestion. Data summarization and application-specific selection and filtering, discard uninteresting information before it is transmitted, improving network performance, but may lead to poor application performance when the process of data selection is overly restrictive or otherwise in need of calibration. When a system is bandwidth limited, it is of course better to plan which sets are lost rather than leave it to chance.

Sometimes we know a priori exactly how to design and tune a sensor network application to reduce significantly the energy it consumes transmitting data. More often than not, however, we find that engineering systems need training, tuning, and calibration, and that researchers interested in the data that sensor networks produce do not know exactly the best way to filter it, because they haven't seen such spatially dense data before. Which nodes have interesting data is application- and environment-dependent and time varying. Correlation between the data of neighboring sensor nodes is not always well understood.

Ideally, then, we'd like the best of both worlds from sensor networks. When we are calibrating our algorithms, testing hypotheses, or simply searching for interesting patterns within the data, we like our applications to collect representative raw waveforms to a back end. When we've discovered how best to filter our data, our sensor networks should trans-

*Contact author; other authors in reverse alphabetical order.

fer data processing to the sensor nodes themselves, to return as much interesting information as possible given our bandwidth limitations.

To tune our network, we should have exactly the same filters and classifiers present on the backend as we have on our wireless sensor nodes. When we find the right parameters for our system – the pass-band and appropriate order for a finite impulse response filter, for example – we should have a mechanism to transfer the responsibility of filtering from the back end to deployed nodes. When our network conditions change, responsibility may transfer back to the backend, so that we may calibrate for operation under new constraints. This process may be iterative over the course of an application’s lifetime.

This paper makes three contributions: We have developed the first acoustic collection system for motes that we know of and the first neural spike acquisition application capable of supporting a network of more than two nodes. Second, we have designed the software system on which these applications run. This system includes a processing library for data measurement, classification, filtering and compression; a fast and low-jitter data acquisition system for resource-constrained motes; and a mechanism to activate and control mote and backend processing of signals.

Finally, we demonstrate through experiments the fidelity tradeoffs in bandwidth limited networks. We show that on-line 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 APPLICATIONS AND PLATFORM

Two target applications motivated our work, each of which collects samples data at a high rate. The *Auricle* system reports acoustic data, while the *Neuromote* system [5] detects and reports neural spikes generated, for example, by a rat’s neurons. Both of these applications are instances of the same TinyOS-based [16] software system, which we call *VanGo*.

Mote-like devices, including the relatively advanced TelosB mote we use, can’t continuously transmit at the rates these applications require. Furthermore, different deployments of either system might require different data compression strategies. In *Neuromote*, for example, different types of signals vary by an order of magnitude in frequency; the compression strategy must allow the network user to choose which signal is most interesting. Nevertheless, the advantages of a mote form factor, including size, cost, deployment flexibility, and low environmental impact, remain important for these applications. Our work shows how to bridge the gap: a design and software structure that lets motes collect high-rate data, even given low-rate radios. This section describes our hardware platform and applications in more depth.

2.1 Sensor Platform

VanGo applications use networks of TelosB motes [21]. 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, and a 16-bit MCU running at 8 Mhz with no hardware divide or floating point support. Its 12-bit ADC is theoretically capable of generating 200 kilosamples per second, but practical sampling rates are limited to about 21 kHz due to insufficient communication bandwidth. Even that presumes that data has no accompanying header information, the link protocol has no header, and that the channel is perfect and available all the time; in practice, it is difficult to support raw data transmission at rates above 2 kHz. For reasons of cost and power consumption, TelosB motes lack application-specific DSPs that could compress raw data in sophisticated ways.

2.2 Auricle

Acoustics have heretofore been used by mote-grade platforms solely for the purpose of target localization; acoustic waveforms have always been discarded after being measured but before transmission. We are losing interesting data. For example, macroscopic acoustic observations, enabled by dense deployments of untethered and unobtrusive sensor nodes, could provide scientists with a deeper understanding of wild-life interactions; proposed projects for acoustic collection systems include monitoring West Coast acorn woodpeckers and marmots. Equally important applications exist in other settings, such as building monitoring and smart spaces.

Auricle goes far beyond our prior acoustic sensor systems [26, 28]: it actually collects acoustic data from a network of motes, rather than using that data to trigger. Each deployed node consists of a TelosB mote connected to an amplifier and microphone. Nodes are tasked by, and send processed data back to, a Linux-based microserver. In general, *Auricle* brings the advantages of mote-based sensors—coverage and scale, low power consumption, low cost, and easy deployment—to audio monitoring.

This system is designed to collect raw acoustic waveforms sampled at more than 8 kHz. (For comparison, the normal range of adult human hearing is approximately 20 Hz to 16 kHz.)

2.3 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 EEG (on the order of 10 Hz) to fast ripples (on the order of 100 Hz), which are an indication of the activity of a population of neurons [1]. Single units, otherwise known as 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 [29]. To detect neuron spikes, however, a sampling rate of at least 2 kHz is necessary.

Existing wired electrophysiological techniques prohibit the study of freely behaving and interacting test subjects in an enriched natural and social environment, due to the tethering caused by wires and harnesses. Thus, the Neuromote application, which 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.

2.4 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. The mote’s small form factor and portability are important for Neuromote, but long-term disconnected operation is not.

3 DATA PATH

Sampling at Auricle or Neuromote rates stretches the limits of current mote hardware and software technology, even assuming limited one-hop transmission and no congestion. Conventional interrupt-driven sampling breaks down at high rates, introducing jitter and preventing other processing, so we built a direct memory access driver for TelosB’s analog-to-digital converter. The software stack must support both interactive experimentation and long-term data collection, and both uncompressed sample formats convenient for manipulation and compressed formats designed for transmission. We designed and implemented VanGo, a new stack that supports both Auricle and Neuromote. Its basic abstraction, the *sample set*, efficiently supports sample processing and application-specific format extensions. Each deployed configuration looks like a single filter chain, significantly simplifying control messages (and therefore interactive experimentation) and simplifying the construction of new filters. Figure 1 describes a typical VanGo software configuration; to support tuning and experimentation, the single filter chain spans across two platforms.

3.1 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 have written a driver for this DMA and modified existing TinyOS ADC code to use the DMA to coordinate the transfer of samples from

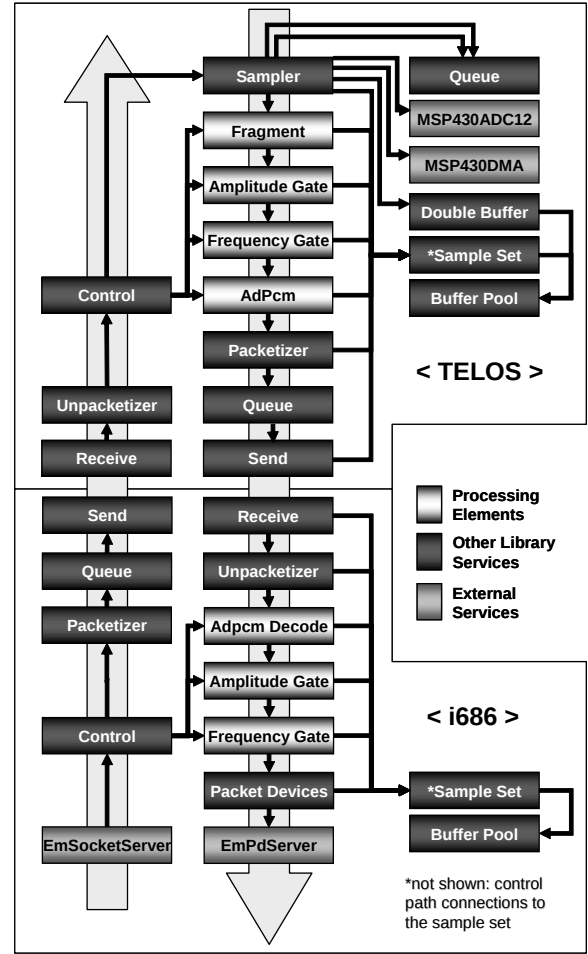


Figure 1—Data and control paths through a VanGo application, including code running on a TelosB mote and on a Linux-based microserver.

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 automatically prefetch a spare buffer that will be available the instant it is requested.

The sampler operates asynchronously, reacting to the 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 the queue becomes non-empty.

In practice, we run the data acquisition subsystem at rates up to 10 kHz. If we do not intend to process or transmit this data, our acquisition subsystem can sustain rates of up to 115 kHz.

For Auricle, the data is generated using a low voltage microphone preamplifier (SSM2167) from Analog Devices powered by TelosB and an omnidirectional condenser microphone (WM-61A) from Panasonic, as suggested by a ref-

erence design from Moteiv. To maintain a constant signal to reference voltage ratio (as our batteries drop in voltage), we use this Vcc as our ADC’s reference voltage as well. For Neuromote, we programmed a Hewlett Packard 33120A arbitrary waveform generator to output a prerecorded neural dataset. The output of the signal generator is AC-coupled to the input of a neural preamplifier circuit, which is an Analog Devices AD627 instrumentation amplifier with its gain set to 200. The neural signals are acquired differentially from the signal generator (as they would be from a live subject), and 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.

3.2 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. 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 sample sets used in our Auricle application have 372-byte sample sets, 68 bytes of which is 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).

Functions are also available for marshalling and unmarshalling sample sets into packet format, facilitating communication with microservers and other PC-class hardware. To make more efficient use of bandwidth, the sample set contains a mask describing which fields to send. 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 runtime, determine that raw data buffers can be twice as large as maximum space available for data in a packet’s payload.

3.3 Filters

The VanGo data path is designed to support flexible signal processing. For example, signal processing elements can attach analysis results to a sample set’s metadata, making it available for later elements to process. In practice, we restrict VanGo-level signal processing to a single linear chain of *filters*, each of which transforms a single input sample set into a single output sample set. This reduces processing gen-

erality, but has several important advantages, particularly for motes. Filters are easy to compose. Performance of a linear filter chain is relatively easy to analyze, which is particularly helpful on hard-to-debug sensor nodes. Control messages are easily directed to the right filter, since each filter appears at most once in the chain. [9]

Not all signal processing algorithms should run as VanGo filters on motes, of course. Many of even the simplest signal processing functions are too CPU-hungry to function in real time on an 8 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, a FFT [24] over a 512-sample window (0.064 seconds of data at 8 kHz runs in 0.5 seconds on TelosB. 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. This accuracy loss adds, of course, to the pressure to adjust processing elements based on deployment needs: recovering the maximum total signal energy from an event requires different compression and gating steps than recovering the most signal from the *closest* sensor to the event (without worrying about other sensors). Section 5 demonstrates this in practice.

We have found that the basic signal processing tasks in our applications can be completed sequentially, where processing functions transform a single input to a single output. Structuring the datapath as a linear chain restricts processing generality, but simplifies composition.

Our filter elements—classifiers, compression algorithms, and measurements—operate in the time domain directly on raw waveform data, as represented by sample set structures. The `Filter` interface type is used to pass sample sets from filter to filter. Each filter has an input and an output interface of this type; the `Sampler` component has a `Filter` output, and the `Packetizer` component (the marshaller) has a `Filter` input. A data path devoid of signal processing functionality appears as like this:

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

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

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

The data processing path of Auricle is:

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

The elements named in such a wiring are the processing elements that will be built into an application. A runtime control mechanism may be used to dynamically enable and disable them individually.

We have written filters to analyze and annotate sample sets, to classify sample sets as being worthy of transmission, and to compress sample sets into more parsimonious formats. The execution times of the filters presented in this paper are summarized in Table 1.

Measurement Filters These filters analyze sample sets, annotating each set with its statistics (using the set’s annotation scratchpad). This accomplishes several goals. Common analysis code is factored out of other filters, which can simply examine the analysis results calculated by a prior measurement filter. Additionally, a filter pipeline might choose to throw out the actual sample data, instead transmitting measured statistics. We have implemented one measurement filter.

The `Stats` filter 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 even multiples of 64 samples each. 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, an indicator of biases in the underlying sensing system. For example, the TelosB ADC assigns the ratio of an input voltage to a reference voltage an unsigned 12-bit value. For acoustic collection and other purely frequency-domain measurements, the center of our waveform should equal half the reference. Small hardware variations lead to imperfections in these measurements; we want to determine the actual signal bias, and do so simply by measuring the signal mean. This is susceptible, of course, to aliasing effects. The components necessary to avoid aliasing would require either additional circuitry, or extensive processing difficult on a mote—an FFT/inverse-FFT pair or a convolution filter requiring more than 30 multiplications per sample. We believe a mote-resident software convolution filter would be possible, but have not yet implemented one. Instead, our experiments deal with frequencies generally below half the sampling rate. The sampling rate is thus above the Nyquist rate, eliminating most aliasing effects; and we observed no aliasing effects in our experiments.

Classification Filters These filters classify each sample set as “interesting” or “not interesting” by modifying an annotation in the sample set metadata. Each sample set begins as interesting; classification filters change uninteresting

sample sets’ annotations to “not interesting,” but leave other sets’ annotations alone. 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. We have implemented three classifiers, two 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. All of these filters use the statistics generated by the `Stats` filter, above.

The `AmplitudeGate` filter marks sample sets uninteresting if they have only low-amplitude data. 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 counts the number of times the signal amplitude crosses the mean and relates this count as a coarse measure of the dominant frequency of the collected signal [27]. Of the known time-domain techniques, this is perhaps the simplest way to estimate a dominant frequency. This gate filters the signal using two exponentially weighted moving averages to perform a simple multi-resolution temporal analysis. A sample set is considered interesting if the fast-moving, fine-resolution EWMA is within a desired dominant frequency range. Each time we transition from not interesting to interesting, we reinitialize the slow-moving, coarse-resolution EWMA to the value of the fast-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 avoiding both false positives and false negatives. We found empirically that for acoustics, smoothness parameters of $\alpha = 0.5$ (fast-moving) and 0.96875 (slow-moving) work well.

Finally, the `SpikeDetector` filter was designed to detect 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 no spikes is not interesting.

Other classifiers are possible, of course, as are other classifier methodologies. For example, filters can mark interesting sets as “interesting” (rather than marking uninteresting sets as “not interesting”); this leads to sets that are interesting if *any* (rather than all) of the classifiers were interested.

Compression Filters Finally, 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 `Format` filter alters the precision and alignment of samples within a buffer. Depending on its parameters, `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, `SpikeDetector` filters out the 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 CONTROL PATH

Motes collect, process, and transmit data. Data reception, backend processing, data presentation, and the creation and dispatch of control messages are the responsibility of the microservers in our network. Hence, our applications are comprised of two executables: one for the deployed mote, another for the backend microserver.

To tightly integrate these two executables, we write all of our services using a combination of `nesC` module definitions [7] and `SNACK` service compositions [10]. In order to execute this code on a PC, we use the `EmTOS` environment [8]. From the perspective of a mote, an `EmTOS` application running on a PC-class device appears to be another mote. However, from the perspective of the Linux system on which an `EmTOS` application is running, it appears to be a standard Linux process that may interact with other system processing using IPC.

Using `nesC` as the base module description language for our entire system greatly 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.

Over the course of a user session with the application, she may alternate between invoking classification elements on the back-end microserver and on the motes. Control over where and how processing occurs is determined by a user connected to the microserver. Backend processing is invoked using the same tasking syntax as is used to control deployed motes.

4.1 Tasking and Control

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 sent 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
```

and to transfer dominant-frequency gating responsibilities from the microserver to mote 4, for instance, we can write:

```
local: dominant-frequency-enable false
4: dominant-frequency-enable true
```

Commands are translated into a shorter binary format before being sent to the control activation service. At the time of this writing, there are about 50 commands defined in our system.

The sampler and all processing elements are initialized and controlled by a single control activation service. The control activation service dispatches signals for local processing control and marshalls control commands for transmission to deployed nodes. Since processing elements may exist on either the motes or their microservers, this service resides on both.

Commands are collected as type-length-value attributes. Consequently, they are stored in the same scratchpad data structure used by the data path.

In single-hop scenarios these messages are delivered directly to intended recipients (either by unicast or broadcast addressing). In multihop scenarios, for reliability control messages are flooded using the `Drip` [15] 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.

Filter	Execution Time (ms)
ADPCM	8.345
Amplitude Gate	0.628
with summarization	0.638
Dominant Frequency Gate	0.776
with summarization	0.786
Statistics	3.790
Format	0.529
Fragment	1.010
SpikeDetector	0.860
with compression	1.017

Table 1—Worst-case run times of our data processing elements for 152 byte (typical) data buffers. The most MCU-intensive filter (ADPCM) consumes 44% of available cycles when sampling at 8 kHz.

4.2 Iterative Experimentation

Real applications must support interactive tuning. Best effort delivery leads to indiscriminate packet loss; packets containing interesting data are just as likely to be discarded as those containing only noise. High-rate data must be classified and filtered before transmission.

Factory calibration of an application’s data extraction parameters is rarely sufficient:

- The deployment environment may produce unpredictable noise and other distractions that should be filtered from the raw signal.
- Climatic and physical variation impacts RF propagation.
- Network topologies may vary in both the level and uniformity of density, making bandwidth availability unpredictable.
- The users of sensor networks haven’t seen spatially dense data before, so they often don’t a priori how best to filter it.

To support interactive tuning, this processing chain spans from the data acquisition subsystem of a mote, to the data presentation facilities of a microserver. Packetization components obscure the physical break in the processing chain (when data is transmitted and received) to provide a single-chain abstraction.

On motes, data produced and enqueued by our sampler is introduced into the chain. Data exiting the chain is transmitted. For instance, a processing chain configuration may look like:

```
Sampler [Filter] -> Fragment
                  -> Statistics
                  -> AmplitudeGate
                  -> Packetizer;
```

On our microservers, data received from the network is introduced into a processing chain. Data exiting the chain is passed to modules responsible for presentation, either by socket or Emstar device files:

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

Data is passed through the chain one buffer at a time. Buffers each typically hold 100 to 200 samples.

5 RESULTS

This section presents experimental results for the Auricle and Neuromote applications. We demonstrate that it is possible to collect high rate data over low rate radios; that our simple and coarse signal processing subsystem works well; and that through runtime configuration of filtering parameters, we can dramatically improve the effectiveness of our collection system.

Our experiments include tests in indoor and laboratory settings, using single-link and multihop communication services. 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

First, we quantify and characterize the Auricle application using an `AmplitudeGate` classification filter. We deploy 12 motes, a single acoustic data source, and a single sink, and define the system’s goal as collecting as much of the signal as possible at the sink. The deployment models a dense sensor network: the source phenomenon is detected simultaneously at all nodes, although with varying fidelity due to environmental factors. If all nodes that detect a signal report that signal, one might reasonably expect link contention to reduce the signal recovered at the sink. Our performance hypothesis is that filtering the sample stream on the motes can increase the total recovered signal power by reducing link contention. This experiment verifies our system’s sampling, statistics, compression, and transmission functions, and its ability to simultaneously perform basic signal processing operations.

Twelve Auricle motes were situated in a straight line with each pair separated by six feet (Figure 2). 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. A microserver sink was deployed close to the middle of the line, one hop away from every mote. We played two minutes of a recording of former president Jimmy Carter’s “Crisis of Confidence” speech, occasionally interrupted by a cell phone ringing. We measured the total signal energy collected at the sink when the motes were instructed to capture everything, and when the motes were instructed to gate on amplitude; amplitude gating essentially provides us with noise suppression, silence elimination, and basic event detection. We sampled at 4 kHz, which is enough to clearly understand a voice, albeit with a noticeable loss of quality. The voice and ring amplitudes were roughly equal. 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. We varied the amplitude threshold for `AmplitudeGate` from 0 (that is, no gating) to 488 mV

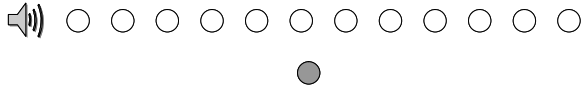


Figure 2—Linear topology used for unicast experiments. White circles represent mote sensor nodes; the grey circle is the microserver sink.

above the mean.¹ The received power was calculated as $P = \sum s_i^2$, where s_i denotes sample i 's AC-coupled value. Only samples collected at the sink were counted. Baseline recordings were conducted to measure the signal power each node is capable of receiving in a contention-free environment, and we normalized our recovered power readings using these baseline measurements.

Figure 3 shows total recovered power as a function of the amplitude gate's threshold. Each data point represents an average of 3 trials.

The shape of this graph is dominated by two competing effects. At very low thresholds, little or none of the signal is filtered before transmission. This results in channel contention and packet loss. Contention becomes less severe as the threshold increases because silent periods, as well as the lower-amplitude signals at distant nodes, are proactively filtered. This, in turn, allows a smaller number of nodes—those better positioned to detect the signal—to have improved access to the wireless channel. 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.

Although the general trend is intuitive, we expected the maximum signal power 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 power 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.

In Figure 3, total recovered power is maximized at a gate setting of around 230 mV. However, this setting does not—say—maximize the closest mote's contribution. Figure 4 plots the per-node signal contribution for the experiments of Figure 3 for three nodes: one close to the signal, one far away, and one in the middle. At very low gating levels, each mote transmits approximately the same number of packets, and thus approximately the same fraction of each node's total theoretical signal power is received. However, since acoustics attenuate rapidly with distance, nodes farther from the source will record the audio at lower absolute amplitudes than closer nodes, and the peak per-node contribution shifts with distance from the source.

¹ 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.

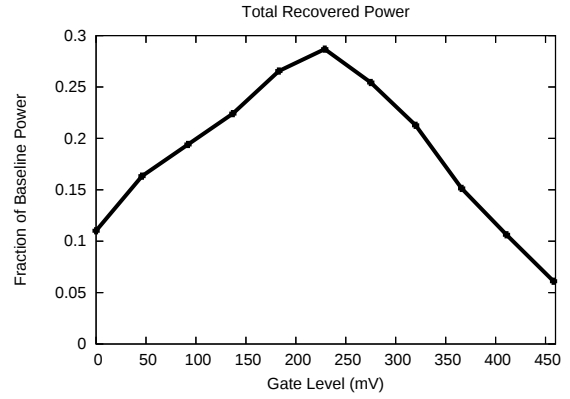


Figure 3—Total power recovered at the sink from all nodes, expressed as the fraction of the power all nodes could send in an error-free, high-bandwidth environment. The peak occurs at a gate setting of 230 mV, which does not coincide with the maximum received power of the mote in the best position to capture the signal (Figure 4). This sharp peak expresses the network-wide point at which congestion effects no longer dominate.

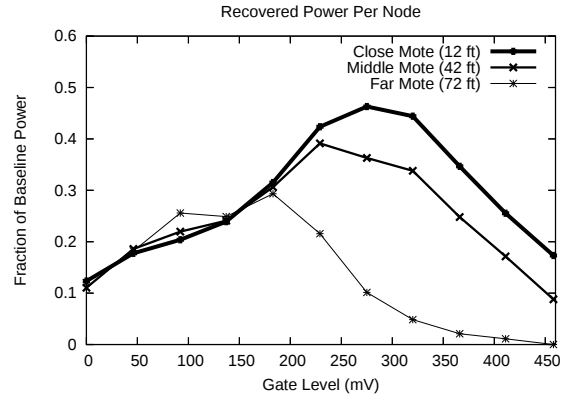


Figure 4—Recovered power for three nodes of varying distances from an audio source, expressed as the fraction of power each node could send in an error-free, high-bandwidth environment. Signal power is maximized by setting the gate threshold just high enough to reduce channel contention. As distance from the audio source increases, peak recovered power occurs at lower gate levels.

Figure 4 shows that closer motes can deliver a larger percentage of signal power than farther motes. Thus, a relatively high gate setting—here, 275 mV, the closest node's peak—will maximize the maximum per-node recovered power. However, this setting is larger than the setting that maximizes the power recovered by the network as a whole (230 mV). It is insufficient, even in the case of a trivial network topology, to use a simple bulk metric to determine filter parameters. Without a priori knowledge of the nature of the signal, the deployment environment, and application usage requirements, the ability to dynamically adjust the parameters of in-network filters is crucial to optimize recovered power.

5.2 Auricle: Filter Composition

Having found an optimal setting for a single classifier, we augment the system with a second, more sophisticated filter: FrequencyGate, which uses a coarse measure of the signal's dominant frequency as a discriminator for classification. We use same the experimental setup as the previ-

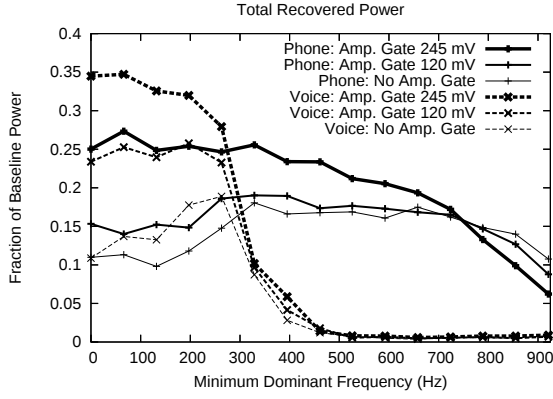


Figure 5—Total signal power recovered at the sink from all nodes, for interesting (“Phone”) and uninteresting (“Voice”) signals, as we vary the minimum dominant frequency to pass. Optimizing parameters for a composition of filters cannot be done by optimizing each filter independently. Recovered power is normalized to the total theoretical power for the relevant portion of the signal.

ous section, but change the goal: now we want to capture as much of the *cell phone* signal as possible at the sink. Our performance hypothesis is that motes are capable of sufficiently sophisticated filtering to discriminate classes of high-rate signals (here, based on frequency). This experiment shows that the dominant frequency filter is effective, far surpassing the amplitude gate as an event detector and classifier for an interesting class of signals. Moreover, we show that filters are as effective in combination as they are alone—more effective, in fact, at reducing signal-to-noise ratio.

We collected baseline measurements as before, and carefully noted the times when the cell phone started and stopped ringing. We use these notes to determine how much recovered power is derived from periods when the cell phone is ringing and, conversely, when the cell phone is not ringing. Since the rings have higher frequency than the voice portion of the signal, we apply our dominant frequency gate to attempt to cut the voice portion of the signal without affecting the phone portion. The `FrequencyGate` classifier is instructed to filter out sample sets whose dominant frequency is below a specified minimum, and 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 240 mV.

Figure 5 shows the results. The frequency filter is clearly effective: a sharp decrease in unwanted (“Voice”) power happens between frequency gates of 200 and 400 Hz, regardless of threshold setting, indicating that the dominant frequency of much of the President’s speech falls in this range. In this experiment, and at the amplitude gates we tested, the combination of frequency and amplitude gating does not greatly improve recovered signal power compared to amplitude gating alone. However, it does improve the signal-to-noise ratio (SNR). The best setting to remove voice while retaining phones lies around 500 kHz, depending on the desired compromise between false negative and false positive readings.

With an amplitude gating level of 245 mV and a dominant frequency filter setting of approximately 525 Hz, the ratio of interesting (ring) power to uninteresting (other) power is approximately 16.6:1, giving an SNR of 12.2 dB. By comparison, when the dominant frequency filter is effectively off, the power 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.

Amplitude gating at 245 mV—approximately the optimal setting determined in the last section—can help or hurt signal recovery depending on the frequency threshold. For frequency gates below approximately 725 Hz, amplitude gating at 245 mV is always better than lower gating levels; above that point, the opposite is true. Such high frequency gates remove enough of the signal that the network is uncongested; additional filtering only reduces received signal power. This again argues that tuning must be done in the field: filter effects and their implications on network performance make iterative, run-time tuning of filter parameters an effective way to improve network performance. Moreover, individual filter elements cannot be tuned individually and then applied together, as such an approach would likely yield sub-optimal results.

5.3 Auricle: Multihop Collection

We now turn from a dense deployment to a sparse multihop deployment: one in which nodes are between one and three hops from the sink, in which sources are heard by relatively few motes, and in which there is more than one source. A single-hop deployment of Auricle already pushes the limits of the TelosB’s resources; multihop networking demands significant additional RAM devoted to forwarding queues, and additional code space to accommodate the routing and transport algorithms. Our performance hypothesis is that the network effects that influence tuning levels in one-hop experiments will only increase in multihop networks, and in particular, that the best tuning levels for recovering signal power will vary based on the distance of a source from the sink. We find that although there is a noticeable performance decrease with multihop networking, the integrated system as a whole operates well. Even in the presence of significant unwanted traffic generators, filter parameters can be tuned to allow a significant portion of theoretical power to be recovered from one node of interest.

Fourteen TelosB motes were deployed in a 3-by-5 grid, with nodes along the length separated by 25 feet and nodes along the width separated by 8 feet. A sink was deployed at one corner of the network, with sources six feet from the nodes at two other corners; see Figure 6. The node closest to one source had one-hop connectivity to the base station, while the node closest to the other had two-hop connectivity. We placed the nodes directly on the ground to increase RF attenuation, thus making the study of multihop effects tenable.² To implement the multihop networking functionality,

²When nodes are placed a meter above the ground, the nominal radio

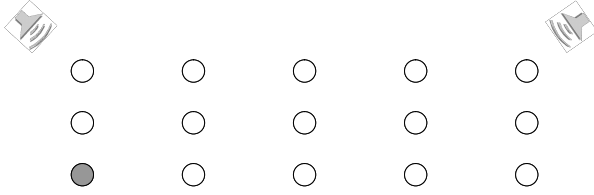


Figure 6—Grid topology used for multihop experiments. Diameter is three hops. The grey shaded node represents the data sink.

we enhanced Auricle to support multihop tree-based collection and to use the MultihopLQI variant of the MintRoute code [30]. In contrast to actively beaconing, as is required to ascertain link quality on more primitive mote radios, MultihopLQI incorporates a value of link quality provided directly by the CC2420 radio into a path cost estimate. To repeat experiments, we froze the routes once they stabilized. To disseminate tasks we use the Drip protocol implementation [15], which reliably disseminates control messages throughout the network with a delay of up to several seconds. As in the single hop experiments, we instruct all nodes to collect data.

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-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 demonstrate the impact of hop distance on optimum gating level. As usual, we compare against baselines measured once per node in the absence of contention—the baselines for the two sources are approximately equal. Each point represents the best result of five trials.

Figures 7 and 8 show the results. The maximum total recovered power, 55% of the theoretical power, is obtained when the 1-hop source group has amplitude gate 275 mV and the 2-hop source group has amplitude gate 450 mV; this point is visible as the 2-hop source's peak in Figure 7. The gating level for the 2-hop source group is probably higher than that for the 1-hop source group because network contention impacts multihop transmission proportionately more than single-hop transmission. This is further visible in the differences between Figure 7 and 8 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 8) achieves less recovered power than it does with a higher gating level in Figure 7. Aggressive, topology-dependent filtering can thus improve the recovered power 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 in a multihop network.

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.

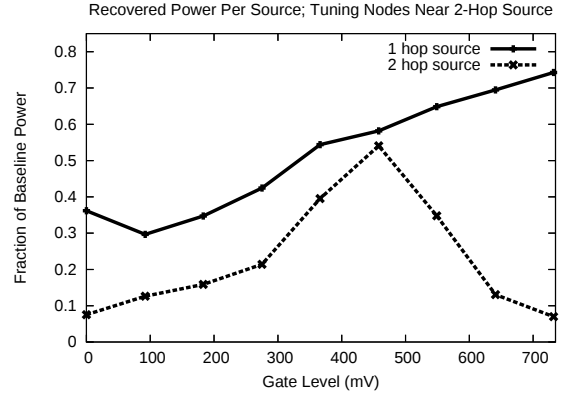


Figure 7—By fixing the gating level at an approximately optimal level of 275 mV for half of the network around the nearby node (1-hop source), and adjusting the gating level for the other half of the network, we achieve approximately 55% of the theoretical power.

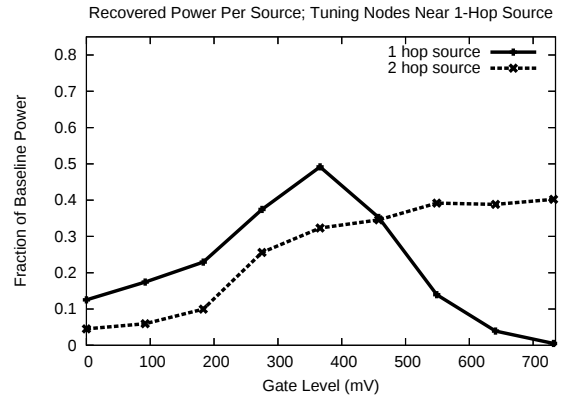


Figure 8—Fixing the gating level at 275 mV for half of the network around the faraway node (2-hop source), and adjusting the gating level for the other half of the network, we achieve at most approximately 40% of the theoretical power. The difference between performance for the nearby and faraway sources is packet drops on bottlenecked forwarding nodes.

5.4 Neuromote

The Neuromote system [5] allows investigators to verify electrode placement and select a spike-filter threshold dynamically by viewing the real-time neural signal. This feature enables Neuromote to be used in a range of test environments (in regard to ambient noise and proximity to neighboring cells). In addition, the investigator can lower the sampling rate remotely to accommodate for a desired network configuration (thus capturing the signals of interest for a desired number of interacting test subjects given a finite network capacity).

In this section we analyze the effects of sampling rate and node density on the detection rate and accuracy of neuron spikes. We show that: (1) The data rate required to transmit complete neuron waveforms of more than one or two nodes will require bandwidth greater than what is supported by the Chipcon CC24420 radio and the supporting software stack. Having more than two nodes simultaneously transmitting complete neural waveforms will result in significant channel contention and indiscriminate loss of waveform data. This loss will increase both with the number of deployed

nodes and the sampling rate. (2) Lowering the sampling rate will lead to post processing error in the measurement of spike heights and widths, and ultimately to missed spikes when we significantly undersample. (3) Since spike signals occupy only about three percent of the raw waveform, the removal of noise between spike events and the transmission of a concatenation of spike waveforms will decrease channel contention.

Our experimental setup consists of an 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 using five four-channel MOSFET input operational amplifiers mounted in the cable connector to remove movement artifacts. Data were recorded wide band (0.1 Hz to 5 kHz) and sampled at 10 kHz/channel (16 channels) with 12-bit precision. 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. The dataset corresponds to one second of neural activity, over which there are seven spikes, each with an amplitude of 1.85 volts (at the ADC input of the TelosB motes), and a peak-trough duration of approximately 1.34ms.

We performed two sets of tests. One test measured the percentage of spikes recovered for different (a) sampling frequencies and (b) number of motes communicating in the network. This test was performed in two modes: raw and spike-only. In the raw mode of operation, the entire sampled signal was transmitted by each node. In the spike-only mode of operation, the spike waveforms were isolated locally by the motes, who transmitted an abridged version of time-referenced spike waveforms only, thus resulting in a decrease in network traffic.

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. However, the the finite resolution (12-bit), and sampling rate (10 kHz) of the original neural signal acquisition apparatus has resulted in a deviation of 108 mV in the dataset that has been programmed into the waveform generator.

Figure 9 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

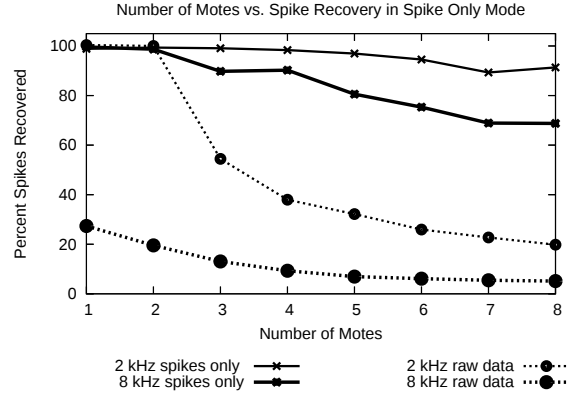


Figure 9—Percentage of recovered spikes as a function of the number of nodes in the network. In the continuous signal transmission mode, 100% of spikes are only recovered with up to 2 motes, both of which must be sampling at 2 kHz. However, when only the waveform containing spike information is transmitted, our spike recovery rate is near 100% for up to six motes sampling at 2Khz and always above 65% when sampling at 8 kHz

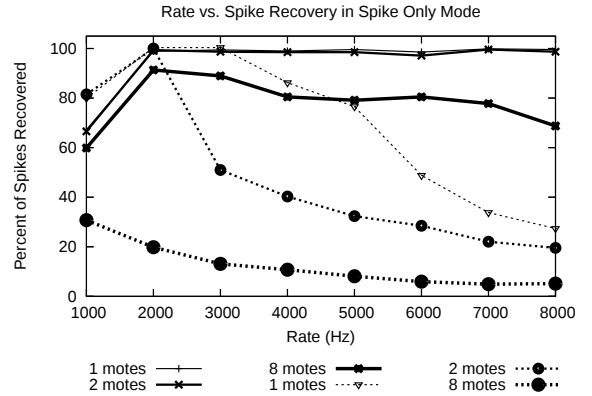


Figure 10—Percentage of recovered spikes as a function of sampling rate and node count in both continuous signal and isolated spike waveform transmission modes. In mote modes, spike loss is observed below 2 kHz due to under-sampling. At 2 kHz and above, due to channel contention the spike recovery rate drops with rate, but much less significantly with abridged data. Packet loss is worse for larger networks.

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 contentions and the spike recovery rate decays linearly from 100% with two motes to 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 8Khz) is roughly two and a half times greater than the maximum for our raw data experiments (with 2 motes at 2 kHz).

Figure 10 describes the effects of sampling rates on net-

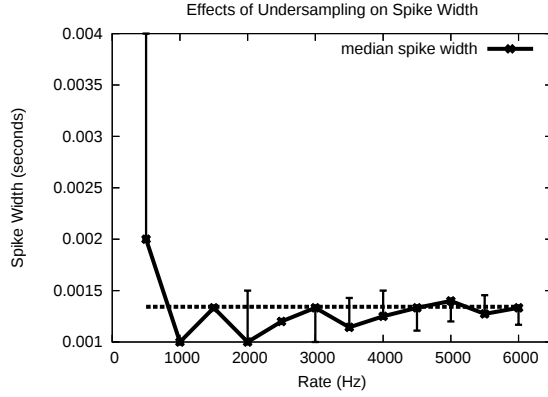


Figure 11—Median spike width as a function of sampling rate. The error bars indicate the 1st and 3rd quartiles. The dotted line at 0.0013 seconds represents the median width for all spikes in the input. At frequencies below 2 kHz, the variation in the recovered spike widths is high due to undersampling. 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.

works 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. But again in this figure, it is evident 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 10 also shows us that there is a penalty to requesting all raw waveforms, due to network congestion. This penalty is more pronounced 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% percent of the spikes when running eight motes in this mode.

The results for the test measuring the variation of acquired spike width and height as a result of different sampling rates are displayed in Figures 11 and 12, respectively. 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 and levels off between 2 and 3 kHz for both height and width variation. At 3 kHz the variation of the spike heights and widths approach those of the original dataset. The variation of spike heights and widths acquired by the motes at a 10 kHz sampling rate are not zero due to slight distortion in the preamplifier and sigma-delta analog-to-digital converter circuits, and 8-bit signal signal quantization (as opposed to the 12-bit quantization of the original signal).

6 RELATED WORK

The applications we have built fall into two categories: acoustic collection and biological monitoring. In sensor networks, acoustic systems are typically factory-tuned to detect spe-

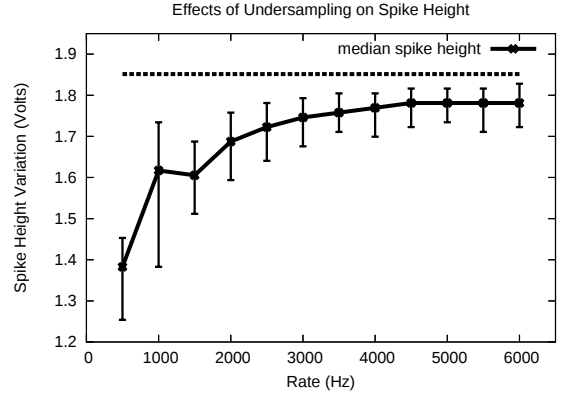


Figure 12—Median spike height as a function of sampling rate. The error bars indicate the 1st and 3rd quartiles. The dotted line at 1.85V represents the actual median height for all spikes in the input. At frequencies increase, the accuracy of the reported spike heights increases substantially. Accurate reporting of spike height enables investigators to classify the cell from which the spike pattern originated with greater precision.

cific events; they transmit information about these events and discard the raw waveform. The counter-sniper system [26], is one example that uses mote networks (and significant complementary signal processing hardware) to detect gunshots and localize their sources. Likewise, VigilNet [11] uses acoustics as part of its surveillance system. Our work considers not only local acquisition of high-rate data and event detection, but also a wide array of data reduction choices to make transmission of the raw waveform tenable.

In the biological domain, the CodeBlue [17] project has developed mote-based sensors with biological interface circuits similar to those used for our neural monitoring application. The CodeBlue sensors are also similar to our work in an application sense, as they are used to acquire and wirelessly transmit biological signals. However, the previously-reported CodeBlue sensor application was not designed to acquire and filter data at the rates necessary, for example, to obtain and transmit neural spike activity.

Until recently, sensor network data reduction has been treated as a collaborative process among motes [13, 18]. Building on the observations of Tenet [9], we diverge from this trend. Although we allow a fairly extensive level of runtime control over signal processing, we constrain motes to process only locally generated data and disallow mote-to-mote collaborative processing. This restriction simplifies our system design. In the imaging domain, Cyclops [23] follows a similar architectural pattern.

The problem of reducing high-rate data to alleviate network congestion has been studied extensively in the context of Internet and cellular transcoding [6, 20, 31]. We share 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. Unlike

the previous work which was focused on the contract between many clients and a single server, we focus on the single client of typical sensor network application and its use of many servers supplying data. Hence, we investigate control across servers. There is some interesting research on distributed music that considers multiple servers providing related data to a client [3, 25] but unlike our project, which focuses on making effective use of bandwidth, these works focused on the difficult task of synchronizing a performance with low latency.

7 FUTURE WORK

The most important next steps for this project focus on the processing library and tasking language. We envision a score of simple waveform classifiers that will detect various types of sounds and sound patterns and report when these sounds occur and measurement components that will augment our statistics component to return basic properties about sensed waveforms in addition to raw waveform data. To support short burst at very high sampling rates, we will implement Flash buffering.

Furthermore, we will expand and formally define the semantics of our tasking language to add features necessary to command motes as groups, focus on motes sensing certain features in their waveforms, duty cycling, adding wakeup events, etc. This work will be done in the larger context of the Tenet architecture.

Real embedded systems need to be optimized. We will be focused on improving the communication subsystems, perhaps adding flow control to multihop transport and windowed acknowledgements to the serial wire protocol.

8 CONCLUSION

We have designed a heterogeneous software system capable of high rate data acquisition, efficient signal processing for data classification and compression, and single- and multiple-hop wireless transmission. We have presented high rate collection applications in two domains: acoustics and biological monitoring. We found that simple filters can impact network performance greatly, particularly in the bandwidth limited environments. As we have shown with both applications, selective filtering before transmission yields more of the data that interests a user. Furthermore, since usage patterns vary and environmental dynamics influence our application behavior, we have found system calibration to require runtime tuning.

ACKNOWLEDGMENTS

This work was made possible by the NSF Center for Embedded Networked Sensing (CENS) under contract number CCR-0120778. SNACK is funded by the National Science Foundation under award number 0435497. We would like to thank Alec Woo and Gilman Tolle for contributing tree-routing code, Phil Levis for the Drip dissemination protocol, and our anonymous reviewers for their valuable feedback.

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 rats with chronic seizures. *Epilepsia*, 40:127–137, February 1999.
- [2] Alberto Cerpa, Jeremy Elson, Deborah Estrin, Lewis Girod, Michael Hamilton, and Jerry Zhao. 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.
- [3] Elaine Chew, Roger Zimmermann, Alexander A. Sawchuk, Christos Papadopoulos, Chris Kyriakakis, Alexandre R. J. Francois, Gerry Kim, and Anja Volk. Musical interaction at a distance: Distributed immersive performance. In *4th Open Workshop of MUSICNETWORK*, 2004.
- [4] ESS. Extensible sensing system: An advanced network design for microclimate sensing. <http://www.cens.ucla.edu>.
- [5] S. Farshchi, P. H. Nuyujukian, A. Pesterev, I. Mody, and J. W. Judy. A TinyOS-based wireless neural sensing archiving and hosting system. In *Proceedings of the 2nd International Conference IEEE Engineering in Medicine and Biology Society Conference on Neural Engineering*, March 2005.
- [6] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-based scalable network services. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 78–91, New York, NY, USA, 1997. ACM Press. ISBN 0-89791-916-5.
- [7] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. 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, San Diego, California, June 2003.
- [8] Lewis Girod, Thanos Stathopoulos, Nithya Ramanathan, Jeremy Elson, Deborah Estrin, Eric Osterweil, and Tom Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *SensSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 201–213, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-879-2.
- [9] Ramesh Govindan, Eddie Kohler, Deborah Estrin, Fang Bian, Krishna Chintalapudi, Om Gnawali, Ramakrishna Gummadi, Sumit Rangwala, and Thanos Stathopoulos. Tenet: An architecture for tiered embedded networks, October 17 2005.
- [10] Benjamin Greenstein, Eddie Kohler, and Deborah Estrin. A sensor network application construction kit (SNACK). In *Proc. 2nd ACM Conference on Embedded Networked Sensor Systems (SensSys 2004)*, pages 69–80, Baltimore, Maryland, November 2004.
- [11] Tian He, Sudha Krishnamurthy, John Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Gang Zhou, Jonathan Hui, and Bruce Krogh. Vigilnet: An integrated sensor network system for energy-efficient surveillance. In *SensSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, New

- York, NY, USA, 2005. ACM Press.
- [12] Crossbow Technology Inc. <http://www.xbow.com/>.
 - [13] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the International Conference on Mobile Computing and Networking*, August 2000.
 - [14] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Next century challenges: mobile networking for Smart Dust. In *Proc. fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278, 1999.
 - [15] P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks, 2004. URL citeseer.ist.psu.edu/levis04trickle.html.
 - [16] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in TinyOS. In *Proc. 1st Symposium on Networked Systems Design and Implementation (NSDI '04)*, pages 1–14, San Francisco, California, March 2004.
 - [17] K. Lorincz, D. J. Malan, T. R. F. Fulford-Jones, A. Nowoj, A. Clavel, V. Shnayder, G. Mainland, , and M. Welsh. Sensor networks for emergency response” challenges and opportunities. *IEEE Pervasive Computing*.
 - [18] Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. In *Proc. 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, Massachusetts, December 2002.
 - [19] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.
 - [20] Steven McCanne, Van Jacobson, and Martin Vetterli. Receiver-driven layered multicast. In *ACM SIGCOMM*, volume 26,4, pages 117–130, New York, August 1996. ACM Press. ISBN 0-89791-790-1. URL citeseer.ist.psu.edu/steven96receiverdriven.html.
 - [21] Joe Polastre, Cory Sharp, and Rob Szewczyk. URL <http://www.moteiv.com>. Moteiv website.
 - [22] Gregory J. Pottie and William J. Kaiser. Embedding the internet: wireless integrated network sensors. 43(5): 51–58, May 2000. URL <http://www.acm.org/pubs/articles/journals/cacm/2000-43-5/p51-pottie/p51%-pottie.pdf>.
 - [23] Mohammad Rahimi, Deborah Estrin, Rick Baer, Henry Uyeno, and Jay Warrior. Cyclops, image sensing and interpretation in wireless networks. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 311–311, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-879-2.
 - [24] Tom Roberts and Malcolm Slaney. Fixed-point fast fourier transform. http://www.jjj.de/fft/int_fft.c.
 - [25] E. Schooler. Distributed music: a foray into networked performance. International Network. Music Festival, Santa Monica, CA, 1993.
 - [26] Gyula Simon, Miklos Maroti, Akos Ledeczi, Gyorgy Balogh, Branislav Kusy, Andras Nadas Gabor Pap, Janos Sallai, and Ken Frampton. Sensor network-based countersniper system. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-879-2.
 - [27] H. Wang, D. Estrin, and L. Girod. Preprocessing in a tiered sensor network for habitat monitoring. *EURASIP JASP: Special Issue on Sensor Networks*, 2003(4):392–401, March 2003.
 - [28] Q.X. Wang, W.P. Chen, R. Zheng, K. Lee, and L. Sha. Acoustic target tracking using tiny wireless sensor devices. In *International Workshop on Information Processing in Sensor Networks (IPSN)*, pages 642–657, April 2003.
 - [29] P. T. Watkins, G. Santhanam, K. V. Shenoy, and R. R. Harrison. Validation of adaptive threshold spike detector for neural recording. In *Proc. or the 26th International Conf. of the IEEE EMBS*, pages 4079–4083, 2004.
 - [30] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *Proc. First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003)*, 2003.
 - [31] D. Wu, A. Swan, and L. A. Rowe. Internet Mbone broadcast management system. In *Proc. ACM SIGMultimedia Conference on Multimedia Computing and Networking* , 1999.