Volcano Monitoring : Addressing Data Quality Through Iterative Deployment

Geoffrey Werner Challen and Matt Welsh

Abstract
Deploying wireless sensor networks to support geophysics presents an interesting challenge. High data-rates required by geophysical instrumentation preclude continuous data collection from even moderately-sized networks. However, geoscientists are used to working directly with complete signals, and therefor uncomfortable with in-network processing that could reduce bandwidth by reporting data products.

Over five years of working with seismologists we have developed a lineage of solutions driven by their scientific goals. Three field deployments have provided valuable lessons and helped drive each successive design iteration. We began by addressing datum quality, encompassing per-sample resolution, accuracy, and time synchronization. Later deployments focused on holistic data quality, which requires considering constraints limiting full data collection in order to maximize the value of the limited data retrieved. This chapter uses our three deployments to demonstrate the benefits of iteration. The first two illustrate our work on datum quality, while the last presents a new approach to optimizing overall dataset quality.

Geoffrey Werner Challen
Harvard School of Engineering and Applied Sciences, e-mail: challen@eecs.harvard.edu

Matt Welsh
Harvard School of Engineering and Applied Sciences, e-mail: mdw@eecs.harvard.edu

1 Introduction

Beginning in 2004, computer scientists from Harvard University joined forces with seismologists from the University of New Hampshire, the University of North Carolina, and Instituto Geofísico, Escuela Politécnica Nacional, Ecuador, to begin a collaboration aimed at using sensor networks to further the study of active volcanoes. As of early 2009 our collaboration has spanned three successful deployments, multiple scientific publications and generated a large number of interesting ideas to explore. We have been fortunate to be a part of this long-running partnership.

From a simple starting point — a handful of nodes streaming continuous data from a single sensor per node — we have developed a sophisticated resource-aware architecture carefully balancing the value of the data to the application against the network-wide cost of extraction. These design changes were motivated by the science goals, responsive to changing hardware platforms, and driven by experience gained deploying prior iterations. Because each of the three deployments is already well-documented, one of our goals is to illuminate the design process by linking successive artifacts together while maintaining the thread of data quality as an application driver.

From the beginning of our work, providing high quality data has remained a part of our research agenda. This focus emerged out of both the scientific goals, and constraints of the devices we have deployed. Because seismologists are used to processing high-resolution data from multiple stations, wireless sensor networks — while considerable less burdensome than existing instrumentation — must provide data of similar quality before they can be used for scientific study. The scale made possible by rapid deployment promised by augmenting existing seismological instrumentation with wireless sensor network hardware is new, but the existence of data processing techniques means that the requirements are already firmly in place.

Designing wireless sensor network applications in this space has required work to meet some of the data quality requirements while finding ways of creatively relaxing others. Specifically, we have found it necessary to deliver high-resolution data meeting strict timing requirements, but found flexibility in terms of providing a complete data set from every node covering all moments of time.

1.1 Overview of Seismoacoustic Monitoring

Volcanic monitoring has a wide range of goals, related to both scientific studies and hazard monitoring. Figure 1 displays an overview of several instruments that might be used, the signals that they collect and example configurations used during deployments. The type and configuration of the instrumentation depends on the goals of a particular study. Traditionally, dispersed networks of seismographs, which record ground-propagating elastic energy, are utilized to locate, determine the size of, and assess focal mechanisms (source motions) of earthquakes occurring within a volcanic edifice [2]. At least four spatially-distributed seismographs are required to

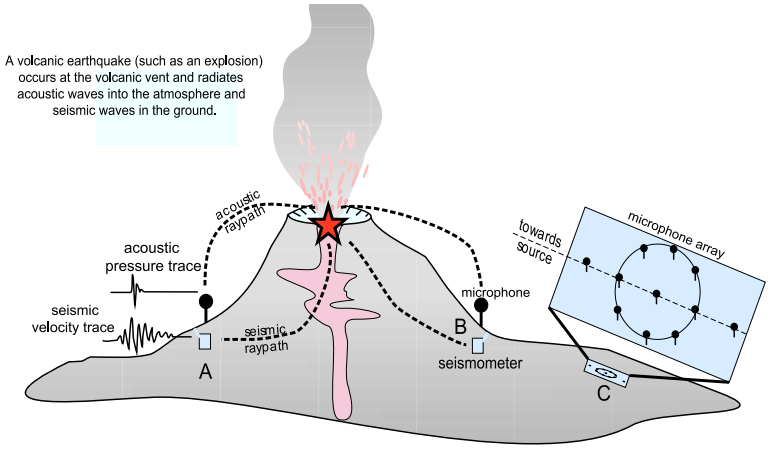


Fig. 1 Sensor arrays for volcanic monitoring.

constrain hypocentral (3D) source location and origin time of an earthquake, though using more seismic elements enhances hypocenter resolution and the understanding of source mechanisms. Understanding spatial and temporal changes in the character of volcanic earthquakes is essential for tracking volcanic activity, as well as predicting eruptions and paroxysmal events [9].

Another use of seismic networks is the imaging of the internal structure of a volcano through tomographic inversion. Earthquakes recorded by spatially-distributed seismometers provide information about propagation velocities between a particular source and receiver. A seismically-active volcano thus allows for three-dimensional imaging of the volcano's velocity structure [1, 15]. The velocity structure can then be related to material properties of the volcano, which may be used to determine the existence of a magma chamber [6, 10]. Dense array configurations, with as many as several dozen seismographs, are also an important focus of volcanic research [3, 12]. Correlated seismic body and surface wave phases can be tracked as they cross the array elements, enabling particle motion and wavefield analysis, source back-azimuth calculations, and enhanced signal-to-noise recovery.

1.2 Opportunities for Wireless Sensor Networks

Networks of spatially-distributed sensors are commonly used to monitor volcanic activity, both for hazard monitoring and scientific research [16]. Typical types of sensing instruments include seismic, acoustic, GPS, tilt-meter, optical thermal, and gas flux. Volcanic sensors range from widely dispersed instrument networks to more confined sensor arrays. An individual sensor station could consist of a single sen-

sor (e.g., seismometer or tilt sensor), or an array of several closely-spaced (10^2 to 10^3 m aperture) wired sensors, perhaps of different types. Multiple stations may be integrated into a larger network installed over an extended azimuthal distribution and radial distance (10^2 to 10^4 m) from the volcanic vent. Data from various stations may be either recorded continuously or as triggered events and the acquisition bandwidth depends upon the specific data stream. For instance, seismic data is often acquired at 24-bit resolution at 100 Hz, while tilt data may be recorded with 12-bit resolution at 1 Hz or less.

Unfortunately, the number of deployed sensors at a given volcano is usually limited by a variety of factors, including: monetary expenses such as sensor, communication, and power costs; logistical concerns related to time and access issues; and archival and telemetry bandwidth constraints. Due to their small size, light weight, and relatively low cost, wireless sensor nodes have an important role to play in augmenting and extending existing seismic instrumentation, providing the increased spatial resolution necessary to support seismic applications like tomography.

Sensor data at a station may be recorded locally or transmitted over long-distance radio or telephone links to an observatory located tens of kilometers from the volcano. At the receiving site, data is displayed on revolving paper helicorders for rapid general interpretation and simultaneously digitized for further processing. However, due to the expense and bandwidth constraints of radio telemetry, high-quality, multichannel data acquisition at a particular volcano is often limited. These analog systems also suffer from signal degradation and communication interference.

As a result, many scientific experiments use a stand-alone data acquisition system at each recording station. The digitizer performs high-resolution analog-to-digital conversion from the wired sensors and stores data on a hard drive or Compact Flash card. However, these systems are cumbersome, power hungry ($\approx$ 10 Watts), and require data to be manually retrieved from the station prior to processing. Depending on the size of the recording media, a station may record several days or weeks' worth of data before it must be serviced. Deploying a wireless sensor network with telemetry to the base station allows real time data collection, network monitoring and retasking not possible with untelemetered systems.

1.3 Overview of Three Deployments

In total, we have performed three deployments of iterations of our system at active volcanoes in Ecuador. Figure 2 shows the location and layout of the second and third deployment. All three are summarized below:

1. **July, 2004, Volcán Tungurahua:** We deployed three infrasonic monitoring nodes continuously transmitting at 102 Hz to a central aggregator node, which relayed the data over a wireless link to the observatory approximately 9 km away. Our network was active from July 20–22, 2004, and collected over 54 hours of infrasonic signals.

2. **August, 2005, Volcán Reventador:** This deployment featured a larger, more capable network consisting of sixteen nodes fitted with seismoacoustic sensors deployed in a 3 km linear array. Collected data was routed over a multi-hop network and over a long-distance radio link to a logging laptop located at the observatory 9 km away from deployment site. Over three weeks the network captured 230 volcanic events.
3. **July, 2007, Volcán Tungurahua:** We returned to Tungurahua Volcano in 2007 and deployed eight sensor nodes in order to test Lance, a framework for optimizing high-resolution signal collection (described Sect. 7). The network was operational for a total of 71 hours, during which time we downloaded 77 MB of raw data.

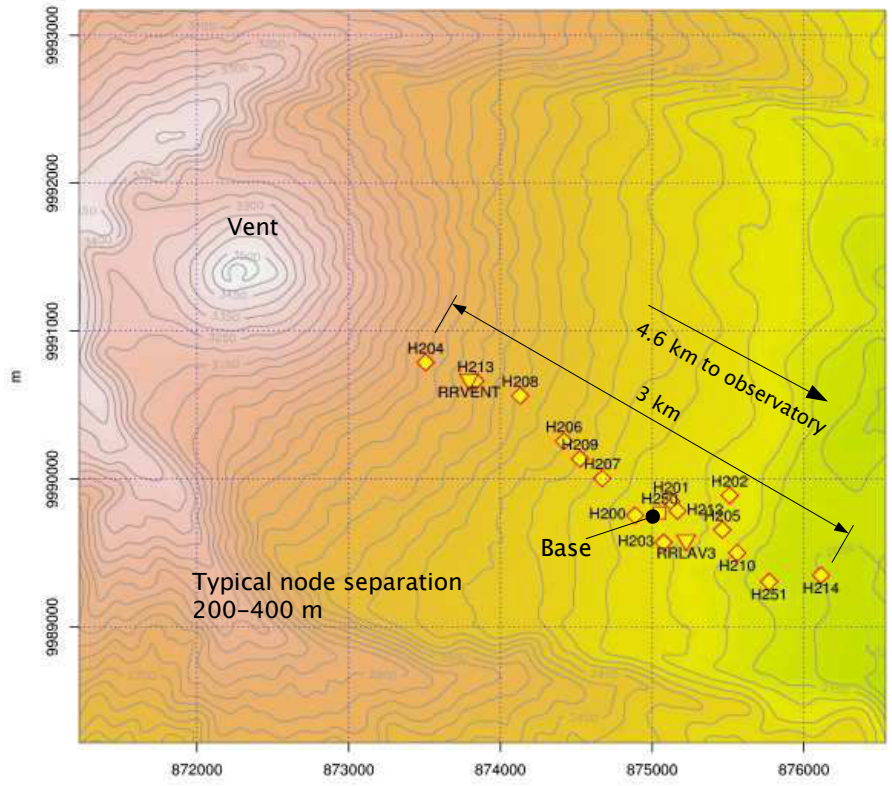
1.4 Datum v Dataset Quality

Reviewing our previous work, we have found it useful to divide our focus on data quality into two separate concerns, *datum* quality and *dataset* quality. The former encompasses the quality of any one data point, and requires addressing sampling rates, resolution, fidelity, and accurate timestamping. Datum quality does not consider broader measures such as the value of the data to the application, coverage or latency. Such higher-level metrics are incorporated in the idea of *dataset* quality, which considers the entire corpus of data presented to the application or end user. This distinction is important because each requires separate techniques to address.

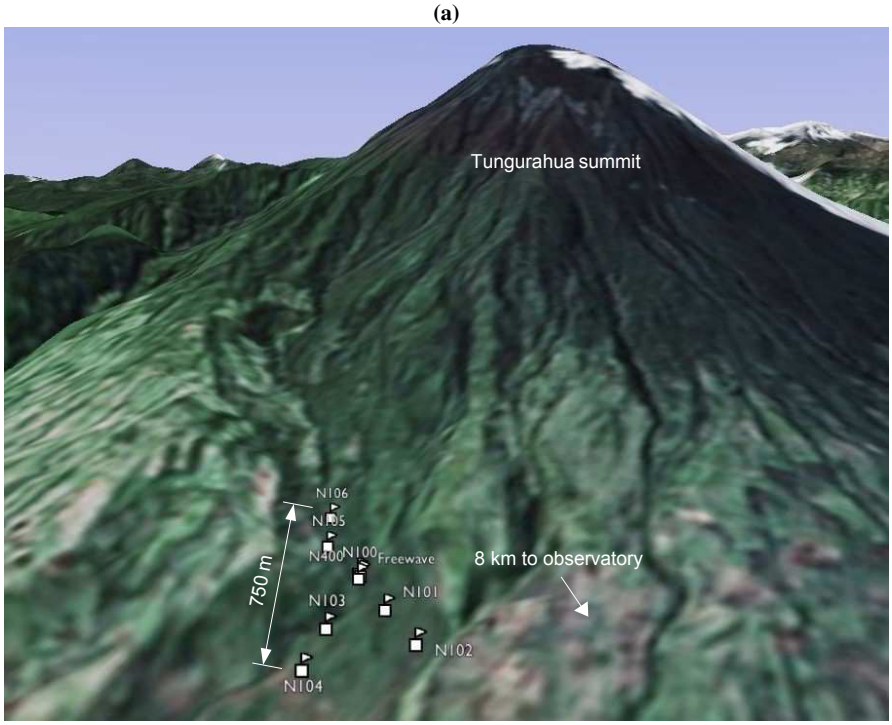
Placed in this context, our first two deployments served to push the datum quality to a level acceptable to the domain scientists. Through the experience that came with iteration, at the end we were able to convince ourselves that we had built a system capable of meeting the science *datum quality* goals [19]. Because the application we chose happened to have fairly well-established datum quality requirements, this work was iterating towards a fixed target.

Our interest in dataset quality reflects the inherent limitations of sensor network devices. Due to storage, bandwidth or power limitations, at some point as the size of the network grows or the target lifetime increases, it becomes infeasible to collect all signals from all nodes during the entire deployment. Thus the question emerges: what data should be collected in real time and what data should not? If the network is provisioned with adequate storage and the deployment is of fixed length, data not delivered in real-time may be eventually recovered manually; otherwise it will be lost.

Return a partial dataset to the end user also requires sorting the wheat from the chaff. Either some of the data has to be interesting enough to justify eliding a great deal of the rest, or some of the data has to be uninteresting enough to justify dropping it entirely. Our application, volcano monitoring, falls more into the first category, since seismologists would never concede that any piece of signal is, prima facie, uninteresting. However, they do have metrics allowing the value of a signal to be estimated and compared against others, which allows system resources to



(a)



(b)

Fig. 2 Deployment locations. Figures **a** and **b** show the locations of our second and third volcano deployments, in 2005 and 2007. Figure **a** shows 17 nodes deployed on Reventador Volcano; **b** shows eight nodes deployed on Tungurahua Volcano.

be directed at the most valuable signals. Because seismological applications may benefit sufficiently from the increased network resolution made possible by small, low-power sensors, the discarded data that these networks imply is tolerable.

The high sample rates and per-sample resolution of seismic signals meant that it only took a medium-size network to make full-signal streaming data collection infeasible. Given that the problem worsens as the network size grows and target lifetimes increase, and our long-term goal is to deploy a perpetually-powered network of several hundred nodes — an order of magnitude larger than any of our efforts to date — addressing this problem remains central to our ongoing research.

1.5 Structure of this Chapter

We begin by describing the key changes made between our initial deployment at Tungurahua Volcano in 2004 and our second more significant effort at Reventador Volcano in 2005. Between these two deployments we made a large number of changes addressing both datum quality — including hardware board development and time synchronization rectification — and dataset quality — including reliable transport and event-driven data collection. We discuss iterative improvements to the interface board and time synchronization in Sects. 2 and 3 below, and outline the event triggered approach in Sect. 4.

After beginning to address the datum quality requirements, we focused on developing techniques that would advance the science goals by allowing the size of the network or the duration of our deployments to be increased. This meant addressing the dataset quality component. Our research in this area has been a great deal more speculative since our seismologists are not used to working with incomplete data sets, and while some of the flaws in earlier systems drove our initial research direction, work in this area became largely driven by the constraints inherent in wireless sensor network nodes: storage, bandwidth, and power. Our work in this area culminated in Lance [17], an architectural framework for optimizing dataset quality in the presence of constraints such as storage, bandwidth and energy. Sections 5 and 7 describe successive iterations of this framework, which emerged as part of our third deployment in 2007, while Sect. 6 describes the policy module component common to both.

Finally, addressing either datum or dataset quality in isolation, even while holding one of them fixed, does not address overall data quality or application-driven quality metrics. Collected data is expected to be put to use in service of some broader scientific goal, which likely has complex data quality requirements not easily distilled into separate datum and dataset goals. We outline future directions intended to bridge this gap and conclude in Sect. 8.

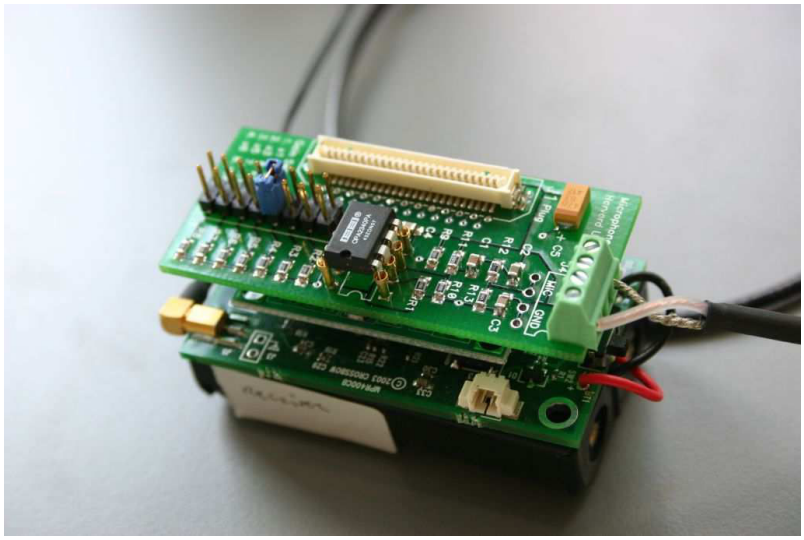


Fig. 3 2004 Volcano Mote. A Mica2 mote with simple sensor interface board allowing a single microphone to be attached as shown.

2 Sensor Interface Board

Of any system component, the hardware interface board received the most significant internal redesign on the way to our second deployment in 2005. Our proof-of-concept deployment deployed nodes fitted with a simple hardware interface board interfacing the Mica2 motes at use at the time with a single microphone. The Mica2's 10 bit ADCs were used to collect acoustic signals at 100 Hz, the sampling rate necessary to capture volcanic infrasound. Sampling was driven by the Mica2's onboard oscillator and the standard TinyOS `Timer` component. Figure 3 shows a Mica2 mote with the hardware interface board attached.

In particular, the following set of limitations of the original board had to be addressed in order to provide the datum quality required by the scientists:

1. **Resolution:** Scientific analysis required a per-sample resolution of at least 16 bits, with greater than 20 being ideal. The Mica2 motes used for the original deployment provided only 10 bits of resolution and later designs, such as the TelosB, provided only 16.
2. **Filtering:** The ideal filter for the seismic and infrasonic data our nodes were collecting is a hard cutoff at 50 Hz. Unfortunately, upon review the original sampling board had a passive filter with a 3 dB point of 5 Hz which meant that desirable components of the signal were being attenuated.
3. **Number of Channels:** Fitting each node with both seismometers and microphones, as we wanted to do for the second deployment, required allowing multi-

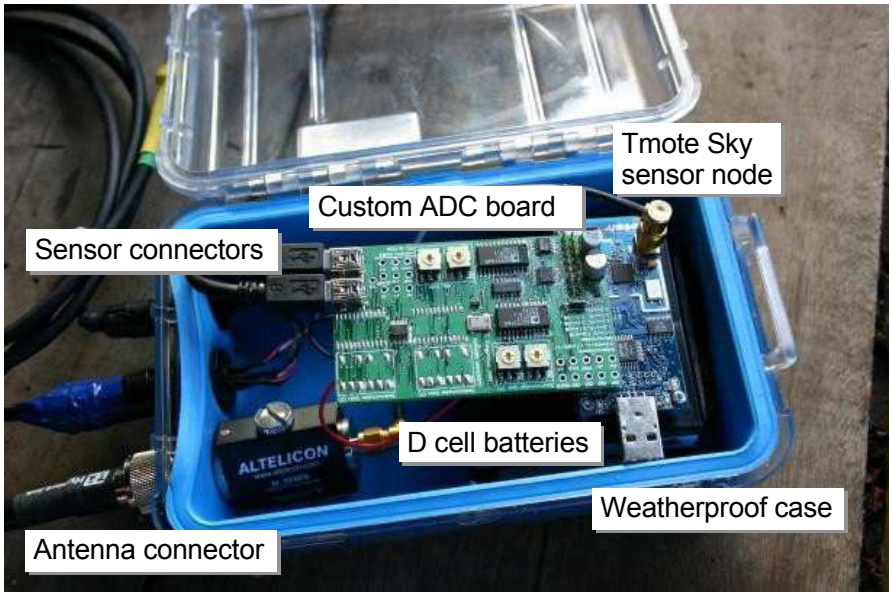


Fig. 4 The second iteration of our volcano monitoring sensor node, fielded at Reventador in 2005 and Tungurahua in 2007.

ple sensors to be attached to each node. Our original board was built to interface a single sensor.

4. **Timing:** We expected that providing the high-fidelity timing necessary to track fast-moving pressure waves across the network would require both a stable timing source driving the per-node sampling and a way of interpreting local times across multiple nodes. Multi-hop time synchronization is discussed separately below, but the timing process necessitated a stable local source that would produce clean per-sample intervals.
5. **Isolation:** When testing our Tungurahua deployment before fielding it we noticed that, due to poor isolation between multiple chips on the Mica2, radio transmissions were interfering with signal collection. The large amount of current required to drive the CC1000 radio was flooding the ground plane and causing the onboard ADC voltage reference to shift. While our original system worked around this crudely in software, providing clean data for scientists would require isolating the sampling components from sources of potential interference.

2.1 2005 Board Redesign

Initially, we considered using the TMote Sky's onboard ADCs. The TMote had a single 16 bit ADC — which we believed to be well-isolated from other circuitry

on the node — and the ability perform DMA sampling from the ADC into onboard RAM while leaving the CPU in a low-power state. However, several limitations of the TMote led us towards a standalone board. First, the 16 bit resolution of the onboard ADC was not quite enough to support our application. Had this been the sole limitation we may have settled for 16 bits, but a separate board meant we could select higher-resolution ADCs. In addition, the TMote had a *single* 16 bit ADC, and we were worried about multiplexing this between multiple channels. Second, we were concerned with providing a stable filter centered at 50 Hz. Building a passive filter at that low of a frequency is difficult, so we concluded that the filtering would have to be done by an oversampling ADC. Performing the filtering on the TMote would have required oversampling at rates it was not capable of.

For these reasons we moved to a standalone sensor interface board providing its own reference voltage (ensuring isolation), ADCs (allowing us to choose the resolution and number of channels and implement digital filtering), and clock source (ensuring a highly-accurate sampling rate). Offloading this much functionality to a hardware board was risky since it left us little room for error in the design and fabrication, but it significantly simplified the code running on the nodes themselves. As it turned out, the fully built-out application we deployed at Reventador Volcano consumed almost all of the TMote Sky's 48 kB of program memory, leaving little room for the functions we offloaded to the interface board.

As deployed in 2005 we were quite happy with the performance of our external sensor interface board. The analysis conducted and reported here [19] showed no significant deficiencies in the board design. There was some initial confusion about the sampling rate which, due to the precision of the oscillator and the settings on the ADC, turned out to be slightly less than exactly 100.000 Hz, but it was extremely consistent across multiple nodes meaning that data collected from several stations could be lined up and processed together. We designed software allowing us to modify the ADC settings on-the-fly and this came in handy during the deployment when we realized that the hardware gain settings were too low. We were able to modify them in software and address the problem without returning to the deployment site.

2.2 Performance and Future Designs

The external interface board was not without its drawbacks. In particular, the oversampling ADCs we chose consumed a large amount of power, around 8 mA per ADC. When combined with other board components, such as the reference voltage, 3 to 5V conversion necessary to power certain components, and external oscillator, the sensor interface board ended up consuming around 60 mA of current, or around 3 times more than the TMote with radio and CPU fully active. As mentioned previously, there is no way to power down the interface board remotely, nor would we be able to without immediately losing data since the ADCs are sampling at tens of kHz. For the Reventador deployment we provisioned around this high power consumption by deploying larger (D cell) batteries and replacing them several times. This can

be seen as a tradeoff between datum quality — which necessitates the high-power external sensor interface board — and a reduction in dataset quality through shorter systems lifetimes or higher-duty cycles due to increased power consumption.

It is likely that the next iteration of this board will take on several additional challenges. First, we will strive to lower the power consumption while maintaining high fidelity through the use of newer ADCs which can hold down their current consumption even while performing the several factors of oversampling necessary to perform digital filtering. Second, we are increasingly interested in the ability to support multiple applications. Thus any future sensor board design, either built in-house or purchased as-is, will be expected to support multiple applications and sensor types. Our current board is, in its choice of ADCs and hardware-filtering, somewhat tailored to the volcano application. We'd like to move away from this if possible.

These two future goals are in many ways at odds with each other, since the challenge of reducing the power consumption of a board designed for a very specific purpose is different and potentially more manageable than the challenge of designing a general-purpose yet low power board. This is a design tension that we expect to continue to play out in future hardware revisions.

3 Time Synchronization

Unlike the sensor interface board, in which a simple prototype led directly to a more successful second effort, achieving high precision time synchronization between nodes is a battle we have continued to fight through multiple design iterations. Indeed, at present we are not yet certain a suitable solution has truly been found, or whether this challenge will reemerge while preparing the next deployment. The problem of accurate timing is one shared by multiple applications and deployment efforts, and significant effort has taken place in this area.

Accurate timestamping arises directly from the analysis performed on seismic and infrasonic volcano data, with the required precision dependent on the intended use and other aspects of datum quality such as the sampling rate. Seismic signals can move across a deployed array at thousands of meters per second, with acoustic signals moving at the speed of sound, (roughly 300 m/s). If two neighboring stations are deployed 100 m apart (possible with good line of sight and powerful antennas) a seismic signal can cross that gap in tens of milliseconds and an acoustic signal in hundreds of milliseconds. At a typical seismological sampling rate of 100 Hz this means that seismic wave arrival times at the two stations might only differ by one or two samples, necessitating precise time synchronization if the propagation of these waves is to be accurately captured. Thus our target accuracy for timestamping has typically been 10 milliseconds: a single sample interval.

Wired seismic instrumentation frequently deploys a single GPS receiver at each station, with the power required to operate GPS-driven timestamping a less significant component of the station's power budget. While we considered this approach

(as described below), we ultimately rejected it due to its prohibitive power consumption.

3.1 Single-Hop Time Synchronization

Our first deployed system made use of a simple time synchronization approach appropriate in a single-hop environment. A single node was attached to a Garmin GPS "puck", which provides a highly-accurate (within 1 microsecond) pulse-per second output. This was trapped by an interrupt pin and, when triggered, that node sent out a broadcast radio message that should reach all other nodes. Upon receiving the message, each sampling node marked the sample that it was in the process of collecting as occurring at that moment in time. Beginning with this mapping between some of the samples (roughly one out of every 100) and the GPS per-second pulse, an accurate timestamp can be assigned to each sample via linear interpolation. (A more complete description of this protocol is contained in previously-published [18] work.)

This simple approach has many desirable properties, particularly when the two enemies of time synchronization in wireless sensor networks — skew and drift — are considered. Skew reflects the fact that all oscillators are not created equal, and that two oscillators sold as identical will, in fact, differ by some small amount. This is particularly true of the less expensive crystals used on low-cost wireless sensor network nodes. Drift names the tendency of the true rate of any particular oscillator to change over time, due to environmental changes such as temperature and humidity. Thus even if two oscillators started out perfectly in synch changes to their local environments would cause them to drift apart slowly over time. Because the GPS PPS is guaranteed to be accurate and displays no drift, the accuracy of the broadcast message rate can be guaranteed. And, even in the presence of skew and drift on the receivers, as long as the drift rates are bounded the interpolation between neighboring PPS signals should yield accurate results. However, the single-hop approach obviously does not work in a multi-hop environment where the multiple sampling nodes cannot all hear the GPS broadcast.

3.2 Adaptation to Multi-Hop Using FTSP

During the three deployments we have performed, a single GPS node was deployed near a large power source (car battery) that also powered other pieces of critical infrastructure (the root of the spanning tree and long-distance point-to-point serial communication linking the deployment site to the volcano observatory). Provisioning multiple GPS nodes with the power necessary to enable acceptable system lifetimes would have greatly increased our deployment burden, and so a software solution was sought.

We ended up deploying a new wireless sensor network protocol called FTSP (Flooding Time-Synchronization Protocol) [8], which was released around the time that we completed our deployment at Tungurahua. FTSP allows a network of nodes deployed into a multi-hop topology to share a single global clock by providing mappings between a local timestamp on any node and the global timebase. Our plan was to timestamp our data by performing two mappings: the first would map the local time on each node into the global FTSP time; the second would map the global FTSP time into the GPS time. We would rely on FTSP to perform the first mapping and deploy a single node with GPS to allow us to perform the second.

3.3 Observed FTSP Instabilities

During the testing that preceded our 2005 deployment at Reventador a number of problems were seen with FTSP in a lab setting. Sometimes the global time would become wildly inaccurate for a period of time before settling back to being quite accurate. We were unable to track down the source of this instability, although we did make multiple changes to the protocol in attempts to harden it and tailor it for our particular application. However, the faults we observed in the lab were all temporary in nature, and we believed that although FTSP did not seem to be always accurate it was stable and able to correct itself when it got off track.

The behavior we observed upon deploying the system was radically different. We did see, periodically, the small, correctable bits of instability that we had observed in the lab. However, we also noticed longer stretches of instability that seemed uncorrectable by FTSP itself. The only solution to rectify the timing on nodes that entered into this state was to reboot them, which forced them to resync upon protocol restart. While we eventually built monitoring and automatic reboots into the system driver software running at the base station, allowing automatic reboots of nodes with troubled timing, the stretches of timing outages frustrated our attempts to collect clean, well-timestamped data.

Figure 5 shows an example of the FTSP instability observed in the field. The global time reported by two nodes suddenly jumps off by several hours, and the nodes do not resynchronize until rebooted 4.5 hours later. It turns out that two bugs combined to cause this problem. First, it was discovered that the TinyOS clock driver would occasionally return bogus local timestamps.[1] Second, FTSP does not check the validity of synchronization messages, so a node reading an incorrect value for its local clock can corrupt the state of other nodes, throwing off the global time calculation.

The failures of the time synchronization protocol made establishing the correct GPS-based timestamp for each data sample extremely challenging. To do so, we developed a *time rectification* approach which filters and remaps recorded timestamps to accurately recover timing despite these failures. Figure 6 shows an overview of

[1] This bug was fixed in February 2006, several months after our deployment.

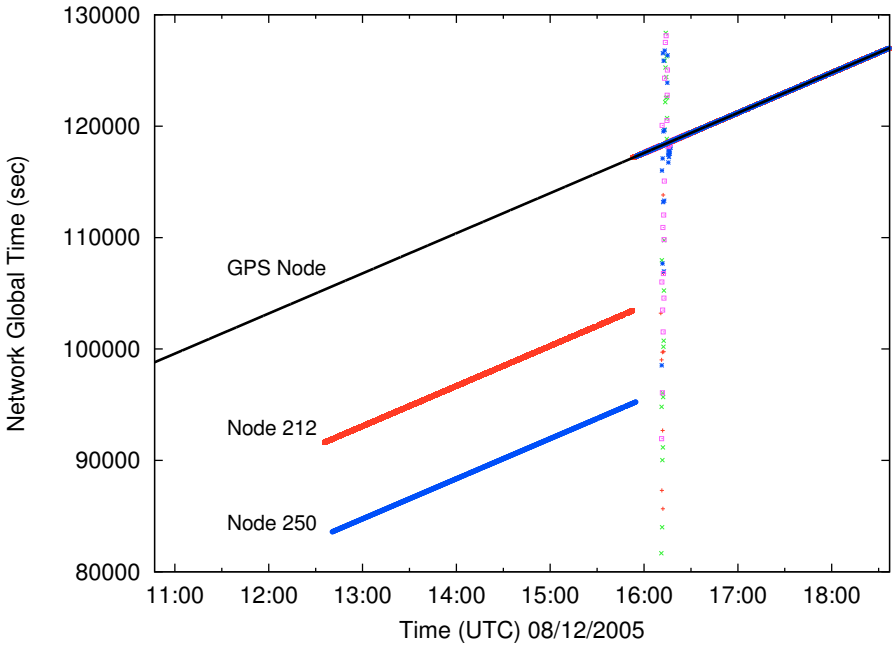


Fig. 5 Example of FTSP instability observed during field deployment. The global time value reported by sensor nodes and the GPS node is plotted against the time that the base station received the corresponding status messages. All nodes are initially synchronized, but starting at 1230 GMT, nodes 212 and 250 report incorrect global times for the next 4.5 hours. When the nodes eventually resynchronize, the global timestamps of other nodes initially experience some instability.

the process. The first step is to *filter* the global timestamps recorded by each node, discarding bogus data. Second, we build a model mapping the local time on each node to FTSP-based global time. Third, we use the GPS timestamp information to build a second model mapping FTSP time to GMT. Finally, both models are applied to the timestamps recorded in each data block producing a GMT time for each sample.

3.3.1 Timestamp Filtering

We begin by filtering out status messages appearing to contain incorrect global timestamps. To do this, we correlate global timestamps from each node against a common reference timebase and reject those that differ by more than some threshold. For this, we use the base station laptop's local time, which is *only* used for filtering FTSP timestamps, not for establishing the correct timing. The filtering process in is many ways similar to prior work [13, 14] on detecting adjustments in network-synchronized clocks.

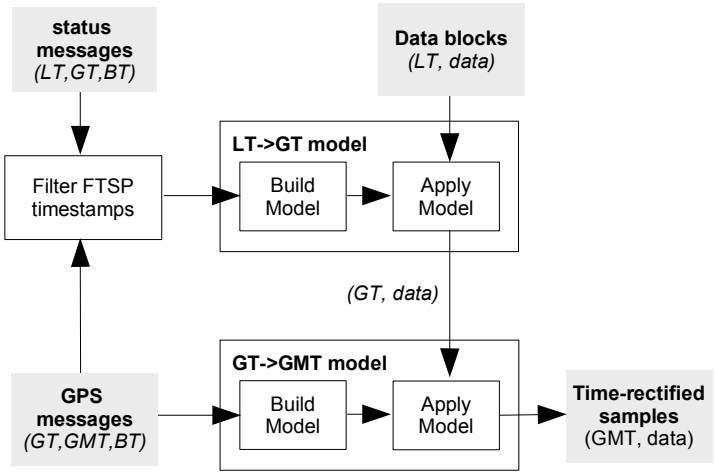


Fig. 6 Time rectification process overview.

We use the following abbreviations: *LT* is the local time of a node; *GT* is the FTSP global time; *BT* is the base station's local time; and *GMT* is the true GMT from the GPS signal. The single GPS node periodically sends a message logged by the base station consisting of the triple *(GT, GMT, BT)*. We use linear regression on this data to produce a reference timebase mapping *BT* to *GT*.[2] Nodes periodically report their status through a heartbeat message, which includes their local (LT) and global (GT) times, and for each node status message logged by the laptop *(LT, GT, BT)*, we map *BT* to the expected GT_{ref} using the reference timebase. If $\mid GT_{ref} - GT \mid > \delta$, we discard the status message from further consideration. We use a threshold of $\delta = 1$ sec. Although radio message propagation and delays on the base station can affect the *BT* for each status message, a small rejection threshold δ makes it unlikely that any truly incorrect FTSP timestamps pass the filter. Indeed, of the 7.8% of timestamps filtered out, the median *GT* error was 8.1 hours.

3.3.2 Timestamp Rectification

The goal of *time rectification* is to assign a GMT timestamp to each sample in the recorded data. In order to do so, we build two models: one mapping a node's local time to global time, and another mapping global time to GMT. Figure 7 shows an example of this process bridging a small local timing instability of the kind described previously.

From those status messages that pass the filter, we build a piecewise linear model mapping *LT* to *GT* using a series of linear regressions. Models are constructed for

[2] We assume that the global time reported by the GPS node is always correct; indeed, the definition of "global time" is the FTSP time reported by the GPS node.

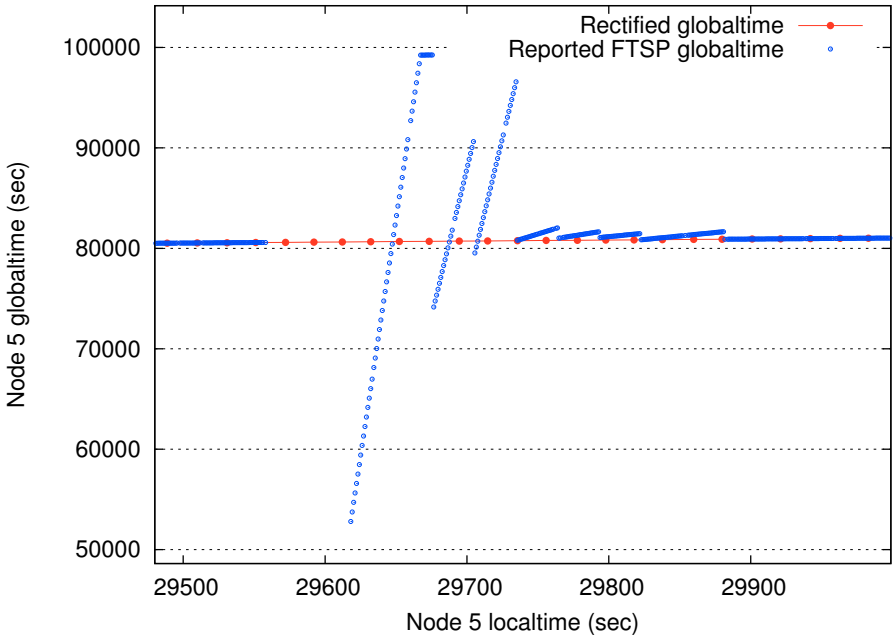


Fig. 7 Time rectification example. The raw (LT, GT) pairs collected from the node show that it experiences a period of FTSP instability. The time rectification process removes the errant timestamps creating an accurate mapping between LT and GT created using a linear regression on the remaining timestamps.

each node separately, since local times vary significantly between nodes. Each regression spans up to 5 minutes of data and we initiate a new regression if the gap between subsequent *(LT, GT)* pairs exceeds 5 minutes. Each interval must contain at least two valid status messages to construct the model. We take the *LT* value stored in each data block and use this model to recover the corresponding *GT* value.

The next step is to map global time to GMT. Each of the GPS node's status messages contain a *(GT, GMT)* pair. As above, we build a piecewise linear model mapping *GT* to *GMT*, and apply this model to the *GT* values for each data block. Finally, we assign a GMT value to each sample contained in the block, using linear interpolation between the GMT values assigned to the first sample in each block. This process makes no assumptions about sampling rate, which varies slightly from node to node due to clock drift.

3.4 Evaluation

Evaluating our time rectification process proved difficult, primarily because we had no ground truth for the timing of the signals recorded in the field. However, by repro-

ducing the deployment conditions in the lab, we were able to measure the accuracy of the recovered timing in a controlled setting. In addition, as described earlier, two GPS-synchronized data loggers were colocated with our sensor network, providing us the opportunity to directly compare our time-rectified signals with those recorded by conventional instrumentation.

3.4.1 Lab Experiments

Our first validation took place in the lab. Feeding the output of a signal generator to both a miniature version of our sensor network and to a Reftek 130 data logger allowed us to directly compare the data between both systems. The miniature network consisted of a single sensor node, routing gateway, and GPS receiver node. The same software was used as in the field deployment. The Reftek 130 logs data to a flash memory card and timestamps each sample using its own GPS receiver.

The results showed a consistent 15 ms offset between the time-rectified signals recorded by the sensor node and the Reftek data logger. We discovered that this offset was due to delays introduced by the digital filtering performed by the ADC on our sensor board (see Sect. 2.1). Adjusting for this delay resulted in an indiscernible offset between the sensor node and Reftek signals. While this experiment does not reproduce the full complexity of our deployed network, it does serve as a baseline for validation.

In the second lab experiment, we set up a network of 7 sensor nodes in a 6-hop linear topology. The topology is enforced by software, but all nodes are within radio range of each other, making it possible to stimulate all nodes simultaneously with a radio message. Each node samples data and sends status messages using the same software as the field deployment. The FTSP root node periodically transmits a beacon message. On reception of the beacon, each node records the FTSP global timestamp of the message reception time (note that reception of the beacon message is not limited by the software-induced topology). Because we expect all nodes to receive this message at the same instant (modulo interrupt latency jitter) we expect the FTSP time recorded at each node to be nearly identical. The FTSP root also records the time that the beacon was transmitted, accounting for MAC delay. The experiment ran for 34 hours, during which time FTSP experienced instabilities similar to those seen during our deployment.

This allows us to compare the *true* global time of each beacon message transmission and the *apparent* global time on each receiving node, both before and after subjecting the data to our time rectification process. We call the difference between the true and apparent times the *timestamp error*. Figure 8 shows the results for nodes one and six hops away from the FTSP root. After rectification, 99.9% of the errors for the one-hop node and 93.1% of the errors for the six-hop node fall within our 10 ms error envelope.

Fig. 8 Timestamp errors in a 6-hop lab testbed. This table shows the 50th and 90th-percentile timing errors on both the raw FTSP timestamps, and rectified timestamps.

	Raw error	Rectified error
1 hop, 50th percentile	1.52 ms	1.42 ms
1 hop, 90th percentile	9.86 ms	6.77 ms
6 hops, 50th percentile	2.63 ms	2.18 ms
6 hops, 90th percentile	13.5 ms	6.8 ms

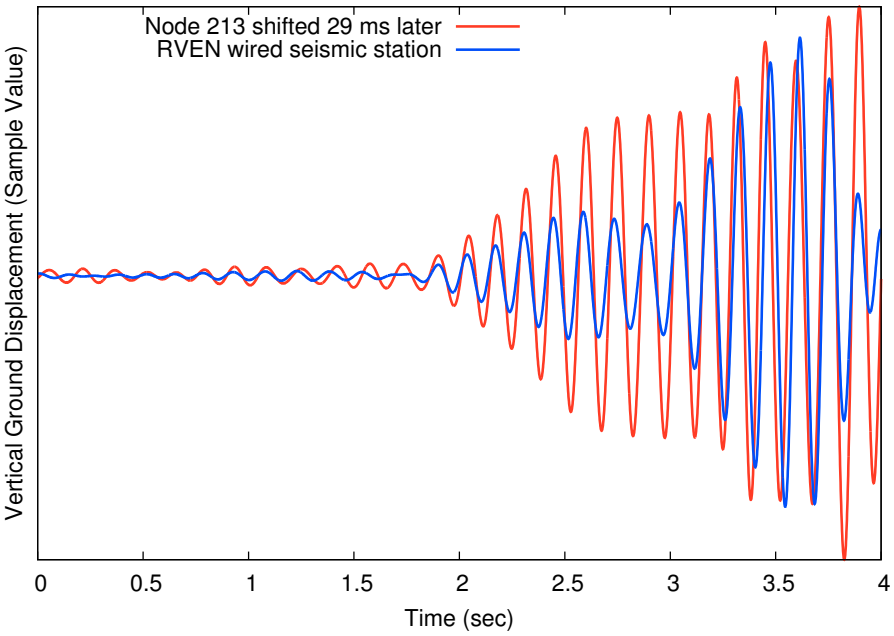


Fig. 9 Comparison of RVEN and node 213 signals. This figure shows two seismic waves recorded by sensor node 213 and a broadband seismometer located 56 m away. After time rectification, a 29 ms time shift produces an excellent match.

3.4.2 Comparison with Broadband Station

Although time rectification works well in the laboratory, it is also necessary to evaluate its accuracy on the data collected during the field deployment. For this purpose, we made use of one of the broadband seismometer stations colocated with our sensor network. The RVEN (for "Reventador vent") station was located 56 m from sensor node 213 (See Fig. 2(b)). Given their proximity, we would expect the seismic waveforms captured by both RVEN and node 213 to be well correlated. Some

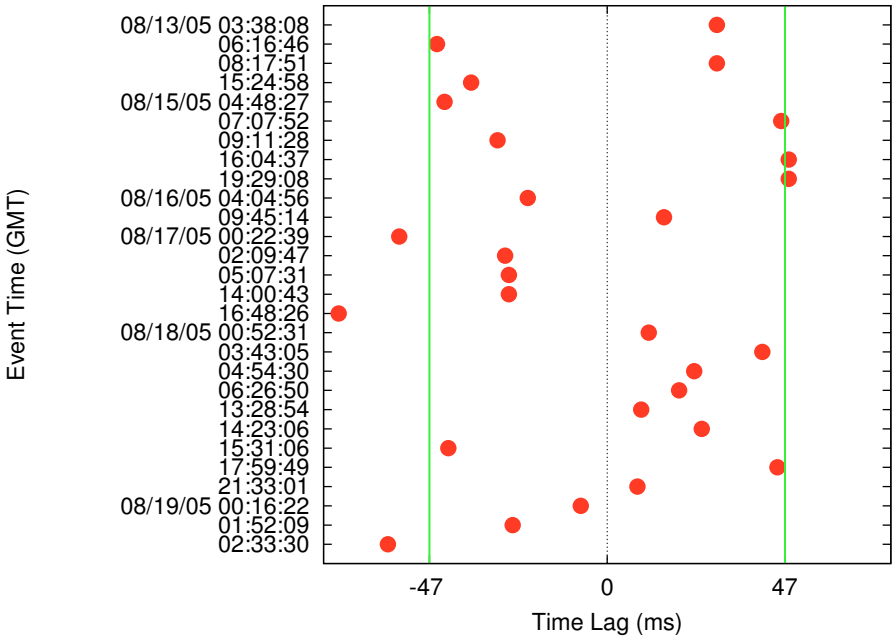


Fig. 10 Lag times between Node 213 and RVEN. The best lag time between the two stations is shown for 28 events. best time lag between the two stations is shown. Most time shifts into the +/- 47 ms window that we would expect given the distance between the two stations and up to 10 ms of timing error.

time shift between the two signals would be expected: a seismic wave passing each station could be as slow as 1.5 km/sec, so the time lag between the signals could be as high as 37 ms. However, due to differences in the seismometers and the placement and ground coupling of the sensors, we would not expect perfectly correlated signals in every case.

We identified 28 events recorded by both RVEN and node 213. The data for node 213 was time rectified as described earlier, and the RVEN data was timestamped by the Reftek's internal GPS receiver. We applied a bandpass filter of 6–8 Hz to each signal to reduce sensor-specific artifacts. The cross-correlation between the signals produces a set of of *lag times* indicating possible time shifts between the two signals. Due to the periodic nature of the signals, this results in several lag times at multiples of the dominant signal period. For each lag time, we visually inspected how well the time-shifted signals overlapped and picked the best match by hand.

Figure 9 shows an example of this process that demonstrates excellent correlation between the RVEN and node 213 signals with a 29 ms time shift. Figure 10 shows a scatter plot of the best lag times for all 28 events. Of these, only 5 events fall outside of a $+/-$ 47 ms window defined by the distance between the stations ($+/-$ 37 ms) and our acceptable sampling error (10 ms). We have high confidence that our time

rectification process was able to recover accurate timing despite failures of the FTSP protocol.

3.5 Lessons Learned

Overall there were many lessons we took away from our experience with data timestamping. First, due to the fact that development of the time rectification technique took several months after the system teardown time, we missed the critical window during which the scientific interest in our data peaked. Arriving in the field with an end-to-end solution ready to take the output of our network and mold it into a form suitable for scientific inspection would have been ideal, and is highly advisable for deployments where scientist have clear datum quality expectations going in, as ours did.

In addition to being unprepared for the issues we observed in the field, we also did not think through the impacts of presenting half-baked data to the seismologists we have been working with. Particularly given our worries about other parts of the system that did end up performing satisfactorily, like the sampling board and driver (Sect. 2), bulk data transfer protocol and event detection mechanism, we were excited when any signals at all showed up at the base station laptop. In our rush to deploy the system we had not prepared an adequate data analysis and visualization environment, and much of this was developed on-the-fly as the deployment progressed. Due to this late and rushed development, none of the initial tools did any of the post-hoc timing rectification that we eventually had to perform in order to make the data suitable for scientific study. Instead, we devised primitive tools allowing data from multiple stations to be plotted together.

Unfortunately, these stacked plots were interpreted very differently by the computer scientists and seismologists. To the computer scientists these were evidence that the network was sampling data, detecting events and successfully retrieving data using our bulk data-transfer protocol, namely things were *working*. Less concerned with the inner workings of our system the seismologists appreciated little of this. To them, these poorly-synchronized signals were instead evidence that our signal timing was extremely broken, a fear that persisted for some time after the deployment as we worked hard together to rectify and validate the timestamps. The exposure of this intermediate data product to consumers unprepared for it should serve as a cautionary tale about how much of the internal engineering of a deployed WSN application to reveal externally.

Validating the timestamps on the data we did collect was also quite frustrating and illustrates to what degree we were unprepared for the severity of the timing challenge. The several wired data loggers deployed alongside our network all had attached GPS units providing extremely accurate signal timing. Had we thought to attach a single sensor both to one of our wireless stations and, by simply splitting the output signal, to one of the wired stations we could have easily cross-checked the timing with a known ground truth (i.e. the identical signal). In a remarkable

oversight we never thought to do this, meaning that we had no two identical inputs available to reveal any inaccuracies in the timing information for the signals we collected. It would also have been useful to spend more time hardening FTSP pre-deployment, and to have built in more logging and visibility into its operation to help us at least have an idea, in situ, of how well it was functioning.

While further work in this area would have been possible, our interests after the 2005 deployment tended more in the direction of improving dataset quality. Given our techniques to rectify the timestamps provided by FTSP were already in place going forward, little additional work was done in this area. In addition, FTSP has continued to be maintained and was canonicized as an official mainline part of TinyOS in version 2.1.1 Given the importance of accurate timestamping to many sensor network applications, certainly ones in the scientific space, it is extremely encouraging that the sensor network community has indicated its willingness to continue to test and maintain this critical component.

4 Event Triggering

While our proof-of-concept deployment had each node attempting to stream a continuous signal over a single radio hop to the base station, this approach did not scale to more nodes deployed into a multi-hop topology. In fact, it didn't even work that well with the small number of nodes we deployed in 2004. Analysis of our data showed many dropouts and periods of missing data caused by simple packet delivery failures. Some nodes were impacted more than others, but all deployed nodes displayed this weakness.

Going into the 2005 deployment we knew that we needed a more scalable approach. The one that we developed was, in its own way, a precursor to the more intensive dataset work that would develop into a separate data-collection framework. What we decided to do was exploit the network's monitoring capability to help decide what data was interesting, and capture that data at the expense of other signals.

The way that this worked was as follows. Nodes were programmed to locally detect interesting seismic events and transmit event reports to the base station. If enough nodes triggered in a short time interval, the base station attempted to download the last 60 sec of data from each node. The download window of 60 sec was chosen to capture the bulk of the eruptive and earthquake events, although many volcanic events can exceed this window (sometimes lasting minutes or hours). To validate our network against existing scientific instrumentation, our network was designed for high-resolution signal collection rather than extensive in-network processing.

Nodes run an *event detection algorithm* that computes two exponentially-weighted moving averages (EWMA) over the input signal with different gain settings. When the ratio between the two EWMAs exceeds a threshold — indicating that the signal's short term average has exceeded its long-term average by a large amount —

the node transmits an event report to the base station. If the base station receives triggers from 30 of the active nodes within a 10 sec window, it considers the event to be well-correlated and initiates data collection.

The bulk-transfer phase operated as follows. The base station waits for 30 sec following an event before iterating through all nodes in the network. The base sends each node a command to temporarily stop sampling, ensuring the event will not be overwritten by subsequent samples. For each of the 206 blocks in the 60 sec window, the base sends a *block request* to the node. The node reads the requested block from flash and transmits the data as a series of 8 packets. After a short timeout the base will issue a repair request to fill in any missing packets from the block. Once all blocks have been received or a timeout occurs, the base station sends the node a command to resume sampling and proceeds to download data from the next node.

At Reventador this event detection approach proved difficult to properly calibrate. We discovered that it detected a small percentage of the seismic events located by one of our domain scientists during a particular window of time. The reason for this was probably the parameters chosen for the EWMA algorithm, which produced a system that was not sensitive enough. Calibrating the system a priori using data collected from the targeted volcano would have been ideal, but was difficult to do given the difference in frequency response between our instruments and the ones that had been deployed at the volcano previously. Other weaknesses of the event-triggered approach to data collection led us to move away from it in subsequent deployments.

5 Addressing Storage and Bandwidth Limitations

The simple event-based triggering we deployed at Reventador volcano in 2005 displayed a number of weaknesses that had to be addressed in order to build a more scalable system. In particular, these limitations led to significant loss of data and the overall approach would not have scaled as the size of the network or the lifetime target increased. After analyzing the performance of the 2005 network we identified three key weaknesses:

1. **Lack of Data Prioritization:**Our event-triggered system attempted to download data corresponding to well-correlated seismic events. However, the event detector operated in a binary fashion, leaving it unable to prefer certain events over others. Because we required each node to make an individual, local decision as to what constituted an “interesting” event, the efficacy of the system as a whole was largely dependent on this parameter. After analyzing the data collected at Reventador we determined that the original threshold was likely set far too low, causing our network to trigger on less than 5% of the actual seismic events observed by wired stations during the deployment. Without the ability to prioritize events setting a threshold means either risking underutilization if the threshold is too high or being unable to distinguish between extremely-interesting and less-interesting triggers if it is too low.

2. **FIFO Storage Management:** Due to the limited flash storage available on each TMote Sky, each node could only store around 20 minutes of continuous sensor data. When an interesting event occurred, to avoid overwriting data about to be requested for download we disabled sampling on each node until the data corresponding to the event in question had been downloaded. Due to the high download latency imposed by reliable transfer over multiple lossy links, and the high event frequency, a large portion of the network was offline for a significant amount of time after each triggered seismic event. This led to data loss.
3. **FIFO, Non-Preemptive Download Policy:** Following each triggered event we downloaded signals from each node in turn until the entire event was captured. As previously mentioned, this download process took a significant amount of time during which many nodes were not sampling. This meant that smaller, less interesting events could prevent the detection of larger, potentially more interesting ones if the small event occurred slightly earlier in time, since the network would still be busy downloading the small event when the large event occurred. Given that many large eruptions are preceded by small precursor earthquakes, this meant that many such large events failed to be recorded at all.

As an initial response to these challenges we developed a utility-driven architecture for optimizing the value of downloaded data in the face of storage and bandwidth constraints. Putting aside the datum quality issues this architecture attempts to address dataset quality directly by allowing the application to express preferences for some data over others and having the system attempt to maximize the value of the downloaded data while meeting constraints on storage, bandwidth, energy and target lifetime. Based on experiences with earlier systems the design of a new system, Lance, was guided by several overarching design principles:

1. **Decouple mechanism from policy.** When possible we wanted to begin to abstract the development of our approaches to dataset problem away from the volcano application context, making Lance suitable for use in multiple application domains.
2. **Simplicity through centralized control.** As previously described, we had significantly simplified the design of our volcano-monitoring network by aggregating functionality on the base station. Lance continues this approach, again treating nodes as slave devices.
3. **Low overhead for maintenance traffic.** The drawback of a centralized solution can be high overhead for the associated control and maintenance traffic. We wanted to avoid this if possible by dividing decision making between local per-node components and global components running at the base station and optimizing the communication patterns between the two.

This architecture developed through two iterations, the second of which was published as Lance [17]. The first and second version of this system share a policy module architecture allowing applications to tailor the data collection strategy. The two iterations differ, however, in significant ways, with most of the changes resulting from our 2007 deployment at Tungurahua Volcano. The first system presents

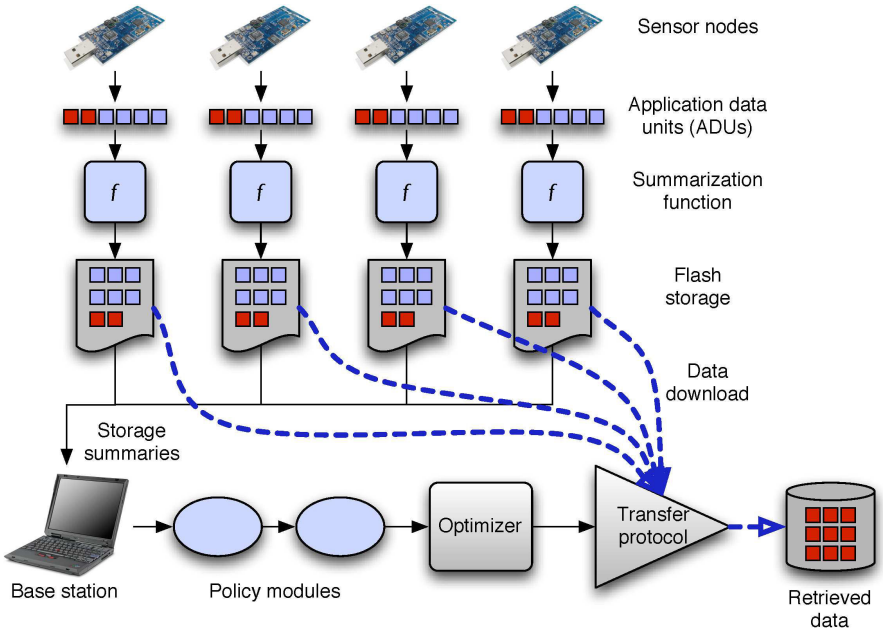


Fig. 11 Lance architecture. The architecture of both the priority- and utility-driven versions of Lance is quite similar. The interpretation of the assigned utility values differs, but the major architectural components are the same.

an *ordinal* (ordering-only) conception of utility, addresses storage and bandwidth as constraints, and attempts to maximize the value of the collected data without considering cost. The second moves to a *cardinal* (ordering and exchange) conception of utility, focuses on energy and bandwidth as constraints, and develops and deploys a cost model as part of the maximization process. This and the two sections that follow place these efforts into the broader context of an approach to dataset quality.

5.1 Overview of Lance

Figure 11 provides an overview of the Lance architecture. Sensor nodes sample sensor data, storing the data to local flash storage. Each application data unit (ADU) consists of some amount of raw sensor data, a unique *ADU identifier*, and a *timestamp* indicating the time that the first sample in the ADU was sampled. ADU timestamps can either be based on local clocks at each node, or tied to a global timebase using a time synchronization protocol such as FTSP [8]. The size of an ADU should be chosen to balance the granularity of data storage and download with the overhead for maintaining the per-ADU metadata. In the applications we have studied,

an ADU stores several seconds or minutes of sensor data, not an individual sample. ADUs are stored locally in flash, which is treated as a circular buffer.

Ideally, nodes would be able to compute the value v_i of an ADU locally, as the data is sampled. However, since the value might depend on factors other than the ADU's data, such as data computed at other nodes. Lance assigns values v_i at the base station, based on global knowledge of the state of the network. However, this requires nodes to communicate some low-bandwidth information on the ADU contents to the base station. For this purpose, each node applies an application-supplied *summarization function*, computing a concise summary s_i of the contents of the ADU as it is sampled. Nodes periodically send *ADU summary* messages to the base station, providing information on the ADUs they have sampled, their summaries, timestamps, and other metadata. As a special case, if a node is able to assign the ADU's initial value directly, this is used as the summary.

The Lance *controller* receives ADU summaries from the network. The controller also estimates the download cost $\bar{c}_i$ for each ADU, based on information on network topology as well as a model of energy consumption for download operations. The ADU summaries and cost are passed through a series of *policy modules*, which provide application-specific logic to assign the value v_i to each ADU. The resulting values are passed to the Lance *download manager* which is responsible for performing downloads, using a reliable data-collection protocol, such as Flush [5].

5.2 Cardinal v Ordinal Utilities

One significant difference between the first and second versions of Lance is the meaning of the utility values assigned to each ADU. Utility values assigned to each ADU are intended to reflect the value of that signal to the application, with higher utilities indicating more valuable data. Accurate and efficient utility assignment – while a difficult and application-specific challenge – is crucial to the Lance approach.

The first version used *ordinal* utilities (or priorities), meaning that the assigned utilities established an order among available ADUs but did not reflect the relative importance of one ADU versus another. Given priorities, an ADU assigned priority 100 can only be assumed to me more valuable that one assigned an ADU 99. It might be 100 times more valuable. It might be only 2 times more valuable. The fact that ordinal utilities produce a strict ordering and nothing else simplified the policies of the first system. The storage manager could easily prioritize available flash on each node, and the download manager simply downloaded the available ADU with the highest utility, irregardless of the resources necessary to do so. Since ordinal utilities do not allow the value of an ADU to be weighed against the cost (in system resources) required to download it, they render the notion of cost unnecessary.

The second iteration of Lance used *cardinal* utilities, which not only produce an ordering but also imply relative value between ADUs. So an ADU assigned utility 100 is assumed to be twice as valuable as one assigned utility 50. This formulation

allows the incorporation of cost into the optimization framework, as described later in Sect. 7.

5.3 Utility Functions

Implicit in Lance is the idea that, when addressing dataset quality, wheat must be separated from chaff. We rely on a utility function to do so, irregardless of whether the utility is interpreted as a cardinal or ordinal one. In general utility functions can be quite complex, depending not only on the data acquired at a particular node but also on the distribution of data across the network, availability of storage, bandwidth, energy and other resources, and so forth. To help designers construct complete systems *without* starting from scratch, Lance attempts to cleanly separate the *policies* governing how to divide data into interesting and uninteresting piles from the *mechanisms* governing the collection of data after the division is performed.

To leverage global knowledge while reducing the resource consumption on each node, we separate local knowledge from global by effectively diving the utility assignments between node-specific (the utility calculator that runs on each node) and network-wide (the policy module chain described in the next section) components. The node-level utility calculator operates only based on inputs available on a single node and must run on a resource-constrained sensor network node, limiting the amount of computation it can perform. In contrast, the network-wide policy modules running on the powered base station have access to a global view of the network and significantly augmented resources. As such they can use global, historical and out-of-band information to further refine the initial utility assignment performed on the node itself.

Communication between the node-level and network-wide portions of the utility function is overhead with respect to the application goals, and so a practical requirement limiting the usage of Lance is keeping the traffic between these two components to a minimum. The utility functions we have developed in the context of the volcano application have typically limited this overhead by having the output of the node-level utility function be a single scalar value, which can be efficiently transmitted to the base station. There is no requirement that the node-level utility function produce such simple outputs, but it does aid in reducing the overhead inherent in the utility assignment process.

As a concrete example tying Lance to previous work, the distributed event detector deployed at Reventador in 2005 could be easily reimplemented in Lance. The node-level utility calculator uses the ratio of two EWMAs to assign a binary utility, either one or zero, to each ADU. At the network level, a policy module aggregates these assignments and, when enough overlap within a particular time window, it marks that entire segment of data with a unit utility value (all other time windows receive utility zero). Because each “interesting” set of signals receives the same utility value, Lance does not distinguish between them and will download them in a LIFO fashion.

Fig. 12 Download performance during the deployment.

Node	ADUs downloaded	Mean throughput
100	311	651.0 B/sec
101	131	446.8 B/sec
102	262	445.8 B/sec
103	292	424.4 B/sec
104	150	256.8 B/sec
105	66	453.7 B/sec
106	20	253.4 B/sec
Total	1232	431.5 B/sec

5.4 2007 Deployment

To evaluate the performance of Lance in a real field setting, we undertook a one week deployment of eight sensor nodes at Tungurahua Volcano, Ecuador, in August 2007. The priority-driven version of Lance was used to manage the bandwidth resources of the sensor network, as described below. Time and budget constraints prevented us from deploying a larger network for longer period of time. Our primary goal was to validate Lance's operation in a field campaign, as well as to identify challenges that only arise in real deployments. Our experiences during this deployment led to further refinements of Lance described in the next section.

The sensor network was operational for a total of 71 hours, out of which the Lance download manager ran for a total of 56 hours. During this time, Lance successfully downloaded 1232 ADUs, or 77 MB of raw data. An additional 308 downloads failed due to timeout or stale summary information, for an overall success rate of 80%. 11012 unique ADU summaries were received from the network, representing an aggregate of 688 MB of sampled data. Lance therefore downloaded approximately 11% of the data produced by the network. Figure 12 summarizes the number of ADUs downloaded and the mean throughput for each node.

5.4.1 RSAM v EWMA Node Level Utility Calculator

The system as initially deployed computed the RSAM [11] (Reduced Seismic Amplitude Measurement) as the value for each ADU. This approach was intended to prioritize data based on the overall level of seismic activity. We experienced two problems as soon as the system was fielded. First, the RSAM calculation was sensitive to DC bias in the seismometer signal, causing Lance to generally prefer downloading ADUs from one or two nodes (those with the largest positive bias). We were able to work around this problem using *policy modules*, described in the next section.

The second problem with the RSAM utility calculator was caused by the uncharacteristically low level of seismicity at the volcano throughout the deployment. We observed only about 20 volcano-tectonic earthquakes and *no* clear explosions, whereas the previous week, Tungurahua exhibited dozens of earthquakes each day. As a result, the RSAM utility calculator was generally unable to distinguish between actual seismic activity and noise. Given the low level of volcanic activity, after the first 25 hours of the deployment we chose to reprogram the network to use a different utility calculator designed to pick out small earthquakes from background noise. This function computes the maximum ratio of two EWMA filters over the seismic signal; it is similar to that described in [19]. Due to code size limitations on the motes, it was necessary to manually reprogram each node with the new utility calculator, which took two teams about 4 hours.

We return to the 2007 deployment later in this chapter in two other contexts. First, we discuss *policy modules*, a useful feature that spanned both the priority- and utility-driven versions of Lance. We illustrate their use with examples from our field deployment. Finally we present the mature, utility-driven Lance, and also include an evaluation of its performance based on data collected during our 2007 deployment.

6 Policy Modules

Policy modules provide an interface through which applications can tune the operation of the download manager. As previously described, along with the node-level utility assignment they make up the utility function responsible for providing input to the download manager. Lance requires that policy modules be efficient in that they can process the stream of ADU summaries received from the network in real time. In practice this is not difficult to accomplish, as the rate of ADU summary reception is modest, and the base station (typically a PC or laptop) is assumed to have adequate resources. For example, a 100-node network with an ADU size of 60 sec would receive an ADU summary every 600 msec. Typical policy modules take a small fraction of this time to run.

One of the main benefits of policy modules is that they permit significant changes to the network's behavior *without requiring changes to the node-level utility assignment*. Changing the latter would typically involve reprogramming sensor nodes. Based on our experience at Reventador we were wary of reprogramming the network unless absolutely necessary. Although systems such as Deluge [4] permit over-the-air reprogramming, any changes to the sensor node software could result in unexpected failures that can be very difficult to debug without manual intervention. On the other hand, introducing new policy modules at the base station is relatively straightforward, and can be quickly reversed without risking sensor node failures.

6.1 Example Policy Modules

Policy modules can be used to encapsulate a wide range of data collection goals, and make it easy to customize Lance's behavior for specific applications. While we provide a standard toolkit of general-purpose policy modules, application developers are free to implement their own modules as well. By composing modules in a linear chain, it is easy to implement various behaviors without requiring a general-purpose "policy language."

Priority thresholding: `filter` is perhaps the simplest example of a policy module that filters out ADUs with a utility below a given threshold T. This type of filtering can be used to force a drop of low-valued data. Conversely, the `boost` utility module sets the utility for an ADU above a given threshold to a very high value, ensuring that it will be downloaded next.

Priority adjustment and noise removal: Policy modules can be used to remove the effects of noise or correct for node-level utility bias, for example, based on poor sensor calibration or differences in site response. Moreover, since each node computes the initial ADU utility based only on local sensor data, it may be necessary to normalize the ADU utilities in order to compare utilities across nodes.

`adjust` adds or subtracts a node-specific offset to each ADU utility in order to correct for differences in sensor calibration. `smooth` applies a simple low-pass filter on the raw utility values to remove spikes caused by spurious sensor noise. Likewise, `debias` is intended to remove sensor-specific DC bias in the utility values assigned by the node's prioritization function. `debias` computes the median utility value for a given node over a given time window. It then subtracts the median from each ADU utility before passing it along to the next module in the chain.

Likewise, when a sensor network contains multiple sensors with varying sensitivity, it is natural to prioritize data from more sensitive instruments. In cases where networks are deployed to monitor fixed physical phenomena, it may be desirable to prioritize data from nodes located close to the phenomena being observed. The `adjust` module can be used to scale raw utilities based on a sensor's location, SNR, or other attributes.

Priority dilation: Another useful policy is to dilate a high (or low) utility value observed in one ADU across different ADUs sampled at different times or different nodes. This can be used to achieve greater spatial or temporal coverage of an interesting signal observed at one or more nodes. The `timespread` detects ADUs with a utility above some threshold T, and assigns the same utility to those ADUs sampled just before and just after.

Likewise, the `spacespread` module groups ADUs from across multiple nodes into time windows and assigns the maximum utility value to all ADUs in that window. Define a window $W(t,\delta)$ as the set of ADUs such that $t-\delta \leq t_i \leq t+\delta$ where t represents the center of the window and δ the window size. `spacespread` determines the maximum ADU in the window $p^* = \arg_{i \in W} \max p_i$ and sets $p'_i = p^*$ for each ADU in W.

Correlated event detection: The `correlated` module is used to select for ADUs that appear to represent a correlated event observed across the entire sensor

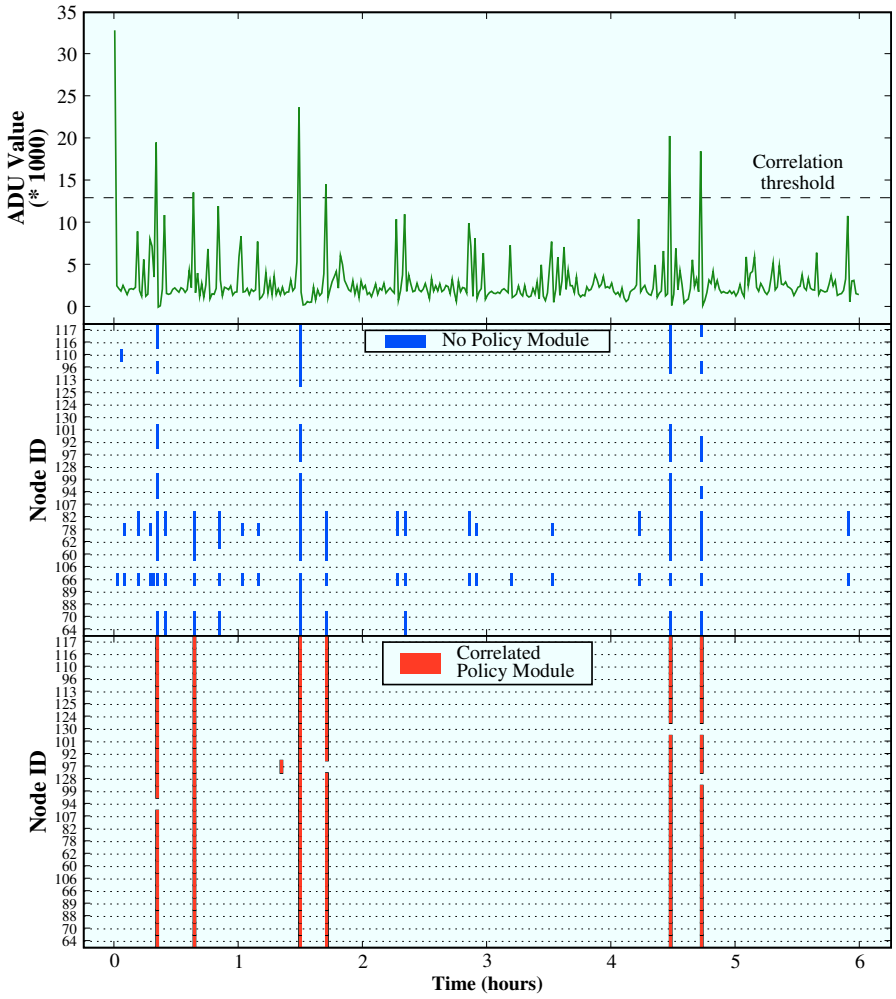


Fig. 13 Usage of policy modules to affect download distribution. Here we illustrate the use of policy modules. The graph compares the download behavior of the system with and without a policy module chain which assigns greater values to ADUs corresponding to correlated seismic activity. The graph is colored at a particular timestamp and node ID if we downloaded that signal from that node. The top graph shows the ADU values over time, with the threshold for the `filter` component of the policy module chain indicated.

network. `correlated` counts the number of ADUs within a time window $W(t, \delta)$ with a nonzero utility value. If at least k ADUs meet this criterion, we assume that there is a correlated stimulus, and the utility values for all ADUs in the set are passed through. Otherwise, we filter out the ADUs in the window by setting $p'_i = 0$ for each ADU in W.

As an example of composing policy modules to implement an interesting behavior, consider the chain

$$\mathit{filter}(T) \rightarrow \mathit{correlated}(k) \rightarrow \mathit{spacespread}$$

This policy filters incoming utilities, rejects time-correlated sets with fewer than k ADUs above the threshold, and assigns the max utility across the set to all ADUs. This can be useful in systems that wish to perform collection of time-correlated data, but avoid spurious high-utility data from just a few nodes. This policy is equivalent to the volcano earthquake detector used in our previous work [19], expressed as a simple policy module chain, and demonstrates, as mentioned in the previous section, that our distributed event detector can be reimplemented as a Lance utility function with both node- and network-level components.

6.2 Evaluation and Use at Tungurahua

We evaluated the usefulness of the policy module architecture through testbed experiments, as well as during our field deployment in 2007. For the testbed experiment, we use a distribution of ADU data values based on a 6-hour seismic trace collected at Reventador Volcano, Ecuador in 2005 [19]. The raw seismic data is divided into ADUs of 36 KB and ADU values v_i are assigned by computing the ratio of two EWMA filters on the data, which assigns greater value to ADUs that contain earthquakes. For each node in a 25 node topology, the ADU values from the seismic trace are attenuated based on a hypothetical signal source and assigned to each of the 25 nodes based on their location with respect to the signal source. We then enable a policy module chain that assigns higher priority to ADUs that correspond to correlated seismic activity across the network.

Figure 13 shows the result of this experiment running on the MoteLab testbed. The upper portion of the figure shows the ADU values over time; the middle portion, the set of ADUs downloaded by the system with no policy modules in use; and the lower portion, the ADUs downloaded with the policy module chain in use. As the figure shows, the policy modules cause the network to prefer correlated seismic events and download an ADU from all nodes in the network when such an event is detected. Gaps in the set of ADUs downloaded are due to download timeouts. In one case, a single ADU is downloaded spuriously due to an incorrect value being reported by that node to the base station. This use of policy modules shows the drastic change in the system behavior that is affected without programming the sensor nodes themselves.

Our deployment at Tungurahua Volcano allowed us to evaluate the ability to change download policies at the base station without reprogramming nodes, one of the significant advantages policy modules provide. As described in Sect. 5.4.1, the RSAM utility calculator initially deployed at Tungurahua was sensitive to DC

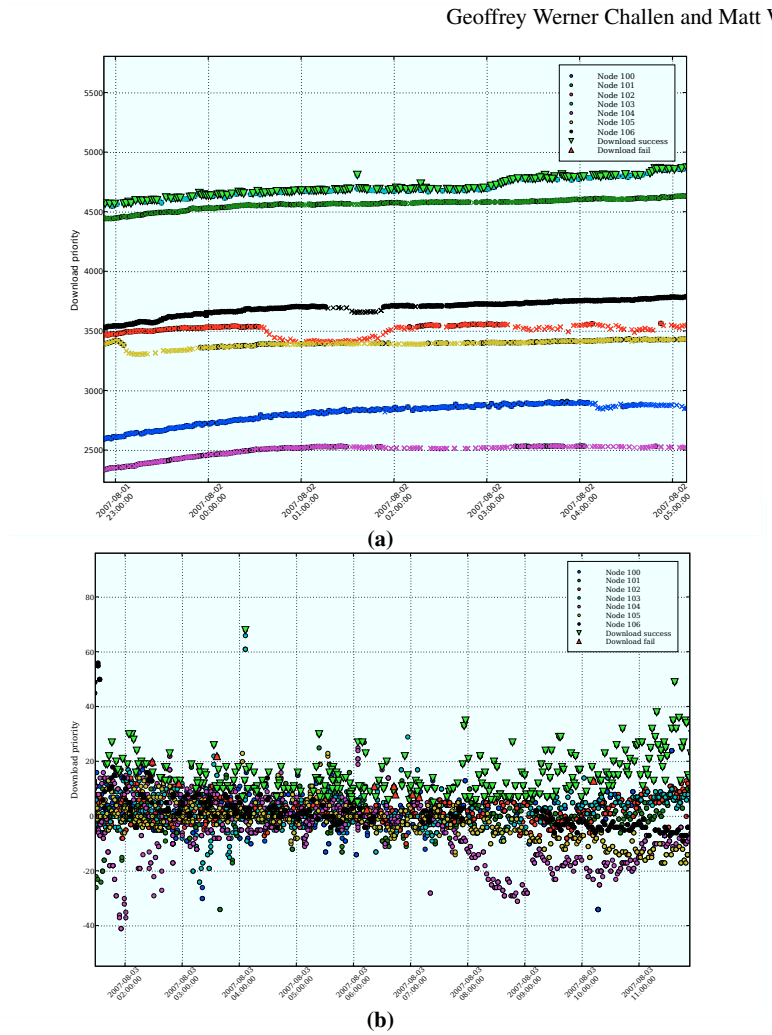


Fig. 14 Effect of DC bias on RSAM summarization function. Each point represents the ADU value received at the base station, and the triangles indicate those ADUs that were downloaded by Lance. In **a**, because nodes' RSAM values are offset significantly from each other, Lance prefers downloading from the node with the largest positive bias. However, **b** shows what happens after applying a simple debiasing policy module at the base station. This filter removes the DC bias causing Lance to download from multiple nodes.

bias, which caused Lance to continuously download signals from one or two nodes with large DC biases.

This problem was easily corrected, without any node software changes, by introducing a policy module at the base station to process the raw RSAM values received from each node and filter out the DC bias. This was achieved by computing the median RSAM value over each 30-minute window of raw RSAM values on each node,

and subtracting the median from the RSAM. Figure 14 shows the result of the application of the filter, with the debiasing effect clearly visible. The ability to change and correct download behavior from the base station without modifying sensor node code proved extremely useful in this case, particularly in that it permitted the rapid iteration necessary to properly craft the appropriate filter.

7 Optimizing for Energy and Bandwidth Usage

After completing our 2007 deployment at Tungurahua we rearchitected our original Lance system and shifted its focus. Instead of optimizing for storage and bandwidth, we instead chose to optimize primarily for energy, with bandwidth as a secondary concern. Several changes to the hardware and software environment explain this shift in emphasis.

First, storage management began to seem less pressing as the promise of sensor network nodes with large attached flash memories seemed to be becoming a reality. The TMote Sky deployed at Reventador had only 1 Mbit of flash, meaning we could only store around 20 min of signal (from two channels, 24 bits per sample at 100 Hz). In contrast, the SHIMMER mote developed for medical monitoring can be fitted with a flash card allowing it to store as much as 2 GB of data, meaning that deployed in support of the volcano application it could conceivable store over 41 *days* of full-resolution sampled data (assuming 2 data channels sampled at 100 Hz with 3 bytes per sample). The idea behind managing storage was that it was necessary in order to preserve the highest-utility ADUs until the network controller would get around to downloading them. Once storage sizes swelled it seemed like a simple FIFO storage policy would allow plenty of time for interesting data to be downloaded.

Second, support for radio duty cycling entered the TinyOS tree and began to be used by our group. This shifted our focus from thinking about bandwidth as a constraint to thinking about the *energy* associated with operating the radio in order to download data as a constraint. In contrast to the early Lance system, which deployed a "greedy" download manager which strove to download as much data from the network as possible, once the radio could be easily disabled when not in use it became natural to want to consider the cost of downloading signals and not necessarily run the downloads in a greedy fashion.

7.1 Refocusing on Energy Usage

Shifting the system's concern to energy usage required several changes to the Lance architecture: a move from ordinal to cardinal utilities, and the development of a cost model.

Replacing the ordinal utilities with cardinal ones allowed us to compare the relative value of different "baskets" of ADUs in a meaningful way. Two ADUs, each with a utility of 50 were considered equivalent to the application as a single one with utility 100. This meant that instead of simply downloading the highest-utility ADU available in the network at a given moment, we needed a way of determining which was the best ADU in terms of both the value but also the system's ability to extract it.

Given our high data rates, we were already restricted by bandwidth alone to downloading only a portion of all of the data sampled by every node in the network. Once we began using LPL to duty cycle the radio extracting data also had an impact on energy consumption as well, and so achieving a given target lifetime could be accomplished by downloading data at slower rates allowing node radios to be powered off when not in use. Either bandwidth or energy could have driven the cost model, but given the dependence of energy consumption on bandwidth usage (and lack of a similar relationship in the other direction) we chose to represent the cost necessary to download an ADU as the *energy* required to retrieve it.

Compared with our original ordinal utility efforts, the new version of Lance contributed several ideas. First was a way of estimating, a priori, the energy necessary to extract an ADU from the network. This was necessary so that both the cost and value of each ADU could be considered before download decisions were made. For value, we leverage the same two-part utility-assignment framework described earlier. The cost estimation component is described below.

In addition, once the cost and value had been assigned to each ADU, the problem of deciding which to download can be framed as an optimization problem. We described a way of producing an optimality bound by mapping an offline version of this problem to a well-studied optimization problem, and use this optimality bound to evaluate several online algorithms suitable for real-time decision making. These contributions are also described in further detail below.

7.2 Cost Estimation

Each ADU has an associated cost $\bar{c}_i$ that represents the energy requirement to download the ADU from the network. $\bar{c}_i$ is a vector $\{c_i^1, c_i^2, \ldots, c_i^n\}$ where c_i^j represents the estimated energy expenditure of node j when ADU i is retrieved. The key idea is that we explicitly model both the energy cost for downloading the ADU from its "host" node n_i and the energy cost for each node along the routing path from n_i to the base station which must forward packets during the transfer. In addition, we also model the energy cost to nodes that overhear transmissions by nodes participating in the transfer. This energy cost on intermediate nodes is non-negligible, since reliable transfer protocols involve a potentially large number of retransmission. However, the overhearing cost is typically small, since modern low-power MAC protocols quickly return to sleep when overhearing transmissions to another node. The cost vector $\bar{c}_i$ therefore depends on the network topology.

Lance estimates the download energy cost vector $\bar{c}_i$ for each ADU sampled by the network. We assume that nodes are organized into a spanning tree topology rooted at the base station. The cost is a function of many factors, including the reliable transport protocol, each node's position in the routing tree, radio link quality characteristics, and the MAC protocol.

Given the complex dynamics that can arise during a sensor network's operation, we opt to use a simple conservative estimate of the energy cost to download an ADU from a node. Our approach is based on an empirical model that captures three primitive energy costs involved in downloading an ADU. The first, E_d, represents the energy used to download an ADU from a given node which includes the energy cost for reading data from flash and sending multiple radio packets (including any retransmissions) to the next hop in the routing tree. The second, E_r, represents the energy cost at intermediate nodes to forward messages during the ADU transfer. The third, E_o, represents the energy cost to nodes that overhear transmissions during a transfer. For simplicity, we assume ADUs of fixed size and compute E_d, E_r, and E_o based on the time necessary to download an ADU from the target node.

Using this simple model, we set the elements of the cost vector $\bar{c}_i$ as follows. $c_i^n = E_d$ for the node n hosting the ADU, and $c_i^m = E_r$ for nodes m along the routing path from n to the base station. We set $c_i^o = E_o$ for nodes that are assumed to be within one radio hop of any of the nodes involved in the transfer. Estimating $\bar{c}_i$ therefore requires knowledge of the current routing topology. This information is readily available: the periodic summary messages, sent to the base station by every node, include the node's radio neighbors and parent in the routing tree. Cost vectors can be easily recomputed whenever the routing topology changes.

To ensure that all nodes meet the lifetime target L, Lance models the energy availability at each node using a token bucket with depth D and fill rate C/L, corresponding to the mean discharge rate. D is determined by the target lifetime L, the battery capacity B and the background drain rate R. In general, $D = B - L * R$, so D represents the energy remaining after the node reserves enough to ensure it can meet its target lifetime at the background level.

7.3 Lance Optimizer

The Lance optimizer is responsible for scheduling ADUs for download, based on knowledge of the set of ADUs currently stored by the network, their associated values, and costs. In our design, Lance attempts to download a single ADU at a time, in order to prevent network congestion, although it may be possible to download multiple ADUs simultaneously, depending on the network topology. A download completes either when the entire ADU has been received or a timeout occurs.

Lance's optimization process attempts to maximize the value of the ADUs retrieved while adhering to the lifetime target L. In essence, we seek a greedy heuristic approximation of the multidimensional knapsack solution that would be used by an oracle with complete knowledge of the ADUs sampled by the network over all

time. The optimizer first excludes ADUs that would involve nodes without enough energy to perform a download. That is, if the token bucket for a given node m has $E(m)$ joules, ADUs for which $E(m) < c_i^m$ are excluded from consideration. Note that as the bucket fills, the ADU may become available for download at a later time. We call these ADUs *infeasible*, and the remaining *feasible*.

To determine the next ADU to download, the optimizer considers the value v_i of each ADU and the its associated cost $\bar{c}_i$. We consider three *scoring functions* that assign a download score to each feasible ADU; the ADU with the highest download score is downloaded next. In the case of ties, an arbitrary ADU is chosen.

The first scoring function, *value-only*, simply downloads the feasible ADU with the highest value v_i. Note that *value-only* will meet the network's lifetime target (since only feasible ADUs are considered) but does not rank ADUs according to cost. The second scoring function, *cost-total*, assigns the score $\hat{v}_i$ by scaling the value of the ADU by its total cost: $\hat{v}_i = v_i / \sum_j c_i^j$. The feasible ADU with the highest score is then downloaded from the network. This approach penalizes ADUs stored deep in the routing tree, which have a higher overall cost than those located near the base station.

The third scoring function, *cost-bottleneck*, scales the ADU value v_i by the cost to the node that is an energy bottleneck for downloading this ADU. That is, let b represent the node with the minimum value of $E(b)$ such that $c_i^b > 0$. *cost-bottleneck* sets the score $\hat{v}_i = v_i / c_i^b$. The intuition behind this scoring function is that the most energy-constrained node should be considered when scoring ADUs for download. We evaluate all three scoring functions in the next section and show that they yield very different results in terms of spatial distribution and energy efficiency.

We evaluate each scoring function against an optimal solution calculated by using knowledge of all future ADUs and solving the multi-dimensional knapsack problem. Since an online solution is required, this provides an upper bound on the achievable performance of the online scoring functions.

7.4 Evaluation and Results

Prior work [17] presents a thorough and detailed evaluation of Lance on a variety of real and synthetic workloads, evaluated both in simulation and in experiments on a large sensor network testbed [20]. We present a subset of these results here to further the discussion.

7.4.1 Simulation and Testbed Experiments

We began by evaluating Lance using a realistic system simulator which allowed parameters – ADU size, distribution of ADU values, network topologies, download speeds, energy costs and target lifetimes to be easily varied. Our simulations focus

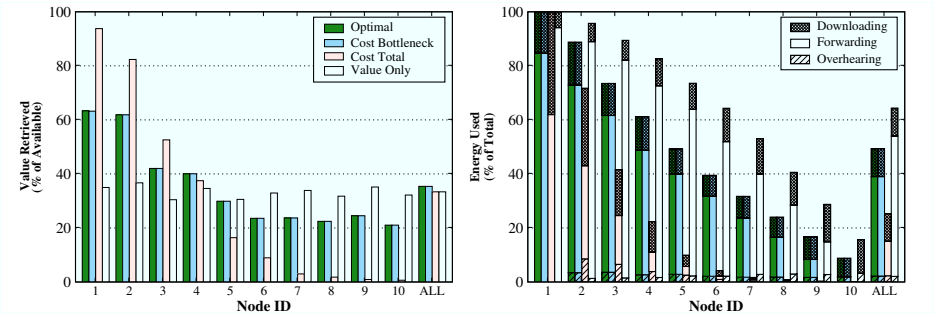


Fig. 15 Per-node distribution of ADU value and energy usage for the linear simulation experiment. The left graph shows the amount of data value downloaded from each node, while the right graph breaks down the amount of energy used by each node into the downloading, routing and overhearing components. Node 1 is closest to the base station.

first on evaluating the candidate scoring functions described above, and secondly on assessing the performance of Lance as the parameters above are varied.

Our first simulation experiment used a simple 10-node linear topology. Figure 15 shows both the performance of each scoring function against the offline optimal system and a breakdown of the estimated energy consumed on each node during the simulation. We divide each node's energy usage into three components: *downloading* energy consumed by the node when it is transferring one of its own ADUs to the base station, *forwarding* energy consumed while transferring data upstream for a downstream node, and *overhearing* energy which accounts for the small overhearing penalty imposed by the TinyOS LPL implementation on nodes that overhear (but are not participating in) nearby transfers.

The energy costs for various operations are modeled as follows. The background current drain of each node is set to 2 mA, based on empirical measurements of a TMote Sky sensor node performing high-data-rate sampling and storing to flash. We also measured the current consumption to download an ADU from a sensor node, and derived the energy costs for downloading ($E_d = 17.6$ mA), routing ($E_r = 16.9$ mA), and overhearing ($E_o = 2$ mA). Our experiments assume that each node can only overhear its parent in the routing tree; developing more detailed overhearing models is the subject of future work. Computing the components of the cost vector for a particular ADU is done by multiplying the current consumption by the ADU download time for each node either downloading, routing, or overhearing the transmission.

Figure 15 confirms the intuition behind the scoring function behavior. *value-only* downloads roughly equal value from each node, but fails to match the optimal performance. *cost-total* downloads more data from nodes near the sink. *cost-bottleneck* comes close to matching the optimal solution, retrieving over 99% of the value retrieved by the optimal solution.

To confirm our intuition we ran a set of simulations using a more realistic 25-node tree topology and using a variety of different ADU value distributions. We

Fig. 16 Optimality of different scoring functions. This table summarizes simulation results evaluating the three different scoring functions. Results are shown for several different lifetime targets and value distributions. *cost-bottleneck* out-performs the others in almost all cases.

		Scoring Functions		
		Value	Cost	Cost
Distribution	Lifetime	Only	Total	Bottleneck
Uniform	4 months	62.4%	90.5%	**93.2%**
	11 months	43.4%	68.0%	**96.9%**
	18 months	44.6%	49.0%	**90.0%**
Exponential	4 months	83.9%	85.1%	**88.0%**
	11 months	70.4%	82.0%	**93.0%**
	18 months	67.2%	72.8%	**91.2%**
Zipfian	4 months	84.7%	**91.4%**	87.1%
	11 months	63.8%	91.1%	**96.2%**
	18 months	53.1%	86.9%	**93.8%**

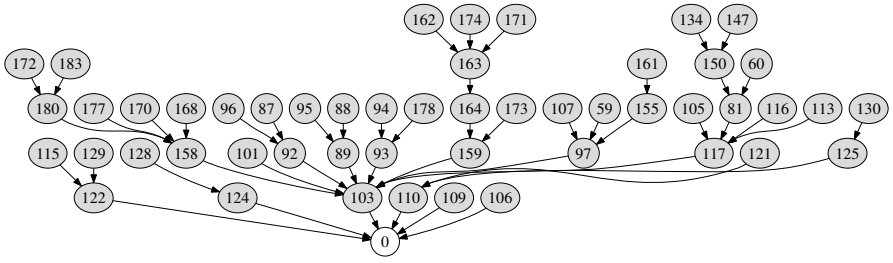


Fig. 17 Topology for testbed experiments. This graph shows the 50 node topology used for the testbed experiments shown in Fig. 18.

draw ADU values from several distributions in an attempt to understand Lance's behavior as the properties of the sampled data change. Three value distributions are used: uniform random, exponentially distributed, and Zipf with exponent $\alpha = 1$. We also make use of an ADU value distribution based on a 6 hour seismic signal collected at Reventador Volcano, Ecuador in 2005 [19] for a later experiment. Table 16 summarizes the results, showing that the *cost-bottleneck* scoring function outperforms the other two in most cases, with optimality values between 87.1% and 96.9%. The one exception is the 4-month Zipfian data set, where *cost-total* slightly outperforms *cost-bottleneck*.

We also ran Lance on our MoteLab [20] Wireless Sensor Network Testbed, in a 50-node configurations shown in Fig. 17. These experiments stress the system in a realistic setting subject to radio interference and congestion, and exercise the multihop routing protocol, Fetch reliable data-collection protocol, and ADU summary

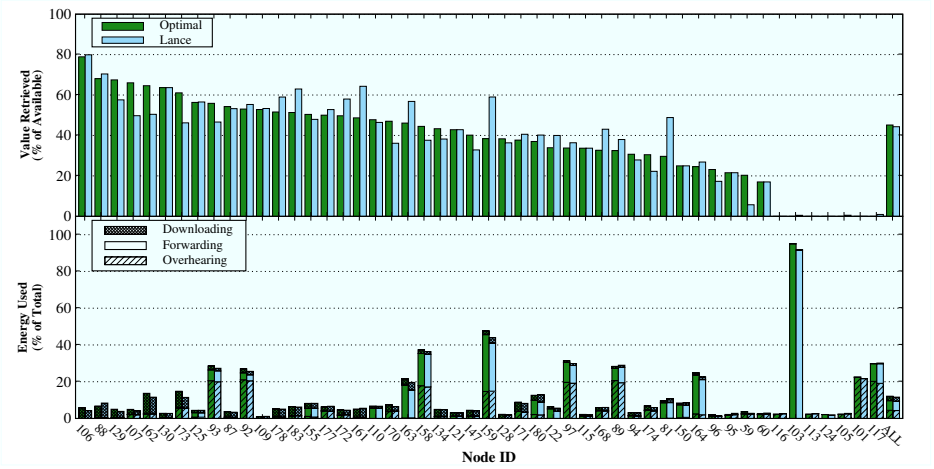


Fig. 18 Optimality and energy use in the 50-node testbed experiment. Lance achieved near-optimal performance during this 8-hour testbed experiment, retrieving 98% of the value obtained by the offline optimal algorithm.

traffic generated by the nodes. For these experiments, we injected artificial ADU values directly into each node rather than relying on the nodes sampling real sensor data; this approach allows us to perform repeatable experiments that explore a wider range of ADU value distributions. We use the *cost-bottleneck* scoring function.

Figure 18 shows the results of a 50-node testbed experiment using a Zipfian data distribution and a target lifetime of 6 months. The upper portion of the figure shows the amount of data value obtained by Lance from each node, compared to the optimal solution (which was computed offline). Nodes are sorted by decreasing optimal value. As the figure shows, Lance achieves very close to the optimal solution, with an optimality of 98% overall. In some cases, Lance incorrectly downloads more data from some nodes and less data from others; this is due to the inherent limitations of an online solution that cannot foresee future ADU values. The lower portion of the figure shows the energy breakdown for each node with downloading, forwarding, and overhearing costs shown. Some nodes consume more than others because of their location in the routing tree. For example, node 103 in uses a great deal of energy for routing packets as it is one hop from the base station, although no ADUs are ever downloaded from that node.

7.4.2 2007 Deployment Analysis

While the earlier, priority-based version of Lance was used to drive our 2007 deployment at Tungurahua, we were still able to analyze this system's performance using utility-based metrics. To evaluate Lance's behavior with respect to an "optimal" system, we took the 8483 RSAM summaries received during a 16-hour period

when the debiasing filter (see Sect. 6.2) was enabled. Using this information, we compute the set of ADUs that the optimal system would have downloaded, with complete knowledge of all ADUs but limited to the same time duration the original network was operating. We assume the download throughput for a given node is always the mean throughput for that node observed during the deployment (see Fig. 12). This calculation ignores energy constraints because the deployed system did not consider energy costs.

An optimal system would have downloaded 392 out of the 8483 ADUs, whereas the actual system downloaded 418 ADUs during this time.[3] The total value of ADUs downloaded by the optimal system is 10678, whereas the value of the actual network was 10629, for an optimality of 99.5%. Lance did an exceptional job of extracting the highest-value data from the network using our online heuristic algorithm.

We can perform a similar analysis during the period of time during which the network was using the EWMA-based summarization function. As with the RSAM-based summarization function, we estimate the optimal set of ADUs that an oracle would have downloaded. During a 25-hour period, the network reported 11012 unique ADU summaries. An optimal system would have downloaded 554 ADUs with total value 577377. The actual network downloaded 518 ADUs with a value of 539115, for an optimality of 93.3%.

As a final evaluation metric, we wish to consider how well Lance, configured in this manner, was able to download seismic signals representing earthquakes. Given the low level of volcanic activity, it turns out that most of the ADUs downloaded by Lance contain no discernible seismic signal. In fact, upon manual inspection of the 518 ADUs downloaded during this period, we identified only 20 ADUs showing a clear earthquake signal, corresponding to only 9 separate seismic events. Note that we did *not* configure Lance to explicitly download correlated earthquakes by using an appropriate policy module (described in Sect. 6), so we would not expect a high degree of coverage for the same event across multiple nodes.

Figure 19 shows the behavior of Lance during a representative 83-minute period. In the figure, we have broken time into windows of one-half an ADU duration (55 sec in this case), and computed the mean ADU value as well as the number of downloaded ADUs that overlap each time window. As the data shows, elevated seismic activity is well-correlated with an increase in the ADU value from across the network, as well as the number of downloaded ADUs. Moreover, the few cases of clear seismic activity in the trace (at times 111000, 112700, and so forth) tend to have more ADUs downloaded. Of the 9 separate seismic events, a total of 27 ADUs were downloaded, representing a per-event "coverage" of 3 ADUs per event. This represents just under half of the 7 nodes participating in the network.

[3] The optimal system would download fewer ADUs than the real system due to the variation in the throughput to each node: the optimal system would download more ADUs from nodes with lower throughput, thereby limiting the total number of ADUs it could download.

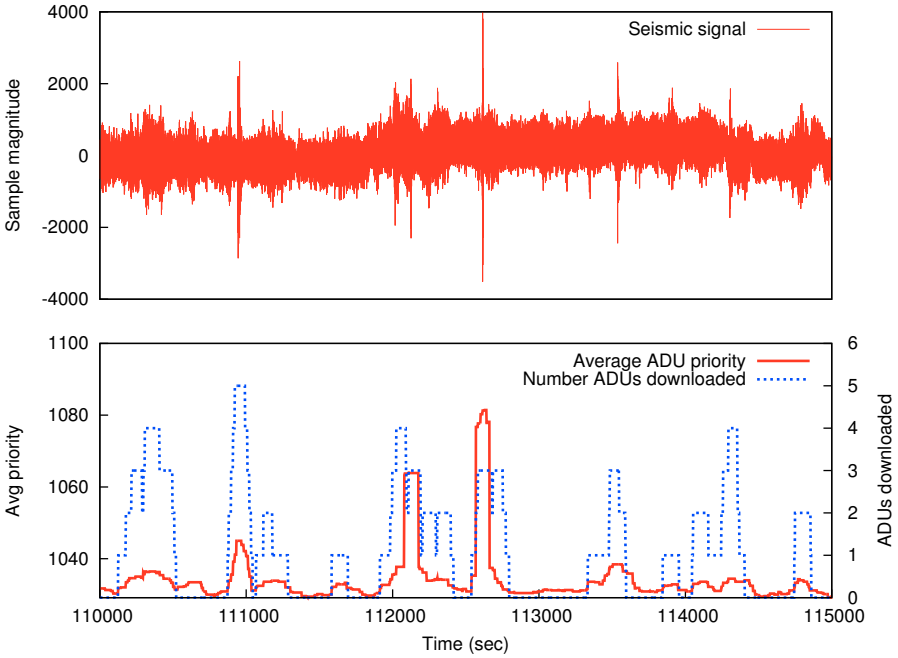


Fig. 19 Lance download behavior overlaid with average ADU value. The top plot shows the continuous seismic signal collected by a single node. The lower plot shows the average value of ADUs and the number of ADUs downloaded for each window.

7.4.3 Results Summary

To summarize the results, we found that the cost bottleneck scoring function allows Lance to approach optimal performance across a variety of network sizes, bandwidth distributions and target lifetimes. By directing scarce energy resources towards the most valuable data, the overall efficiency of the network considered as a whole can be significantly increased.

8 Conclusions and Future Work

Based on our experience with data quality spanning three deployments and multiple system components we see many directions for future work. Every component of datum and dataset quality mentioned here is open for future research and continued development.

While we have made progress in pinning down the the datum quality requirements specific to this domain, work continues on hardware-software co-design, time synchronization and reliable data collection. We are currently designing a sensor in-

terface board for the iMote2 intended to support the volcano application as well as other high data-rate sensing tasks, and are revisiting many of the original design decisions that produced the volcano monitoring board described here. Recent efforts within our group have continued to work on reliable time synchronization in other settings such as body sensor networks. And we have begun a redesign of the data collection component inspired by approaches such as Hop [7] that seek to reduce power consumption and improve performance over lossy, low-power links.

As the Lance work brings out, there are tradeoffs possible between a deployed networks target lifetime and the quantity and quality of the data provided to the application. Lance takes a position on one end of a sliding spectrum in that it holds the system lifetime constant, treating it as an input parameter that must be met, and then tries to turn other knobs to maximize the output data quality given other constraints. Another approach would be to hold the output data quality constant and try to allow the system to last as long as possible while providing data at a minimum fidelity set by the application. Between these two endpoints there are multiple approaches which would tend to trade off application data quality for increased system lifetime. Given the difficulties inherent in performing this tradeoff, we have not yet explored this interstitial area. However, this area seems quite fertile for future work.

In addition, maximizing the output of a deployed network under resource constraints requires continuing to build strong connections between the notion of data quality operative within the network and the actual needs of the application. In Lance, we do not dictate that applications use a simplified scalar score to indicate data quality, but our architecture tends to push applications in this direction. Many applications, however, may not be able to reduce a complex set of considerations into a single scalar value. Enriching this interface may help allow further dataset quality optimizations. In general, however, the interface between the end user's goals and the simplified metrics operating within the network must receive further attention to ensure the broader applicability of solutions that attempt in-network optimization.

In this chapter we have attempted to zero in on data quality issues inherent in the development of a sensor network supporting the monitoring of active volcanoes. Through our experience in this area spanning three deployments we have grappled with many aspects of data quality, from where the sensor meets the sensor node up to complicated optimization approaches running at the base station. We have attempted to narrate, in a helpful way, our experiences in this area, and hope that some of the struggles we have elucidated will smooth the waters for other sensor network deployments.

Acknowledgements The authors would like to acknowledge those who helped make this project possible. Professors Jeff Johnson (New Mexico Tech) and Jonathan Lees (University of North Carolina), our seismology collaborators, made the entire endeavor happen, schooling computer scientists in the fine art of volcanologic research and providing invaluable assistance with the design and deployment of each of our systems. Mario Ruiz (IGEPN) and Omar Marcillo (New Mexico Tech) provided additional domain science and deployment assistance. Harvard students Konrad Lorincz, Pat Swieskowski and Stephen Dawson-Haggerty helped design and implement the sensor network systems we deployed. Researchers, staff and students at the Instítuto Geofisico, Escuela Politécnica

Nacional (IGEPN) in Ecuador provided essential ground support during each of our deployments. Finally, Jim MacArthur and William Walker at the Harvard School of Engineering and Applied Sciences Circuit Lab aided significantly in the design and construction of our interface board and enclosures.

References

1. Benz, J., et al.: Three-dimensional P and S wave velocity structure of Redoubt Volcano, Alaska. J. Geophys. Res. **101**, 8111–8128 (1996)
2. Chouet, B., et al.: Source mechanisms of explosions at Stromboli Volcano, Italy, determined from moment-tensor inversions of very-long-period data. J. Geophys. Res. **108**(B7), 2331 (2003)
3. Dietel, C., et al.: Data summary for dense GEOS array observations of seismic activity associated with magma transport at Kilauea Volcano, Hawaii. Tech. Rep. 89-113, U.S. Geological Survey (1989)
4. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: Proc. 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04) (2004)
5. Kim, S., Fonseca, R., Dutta, P., Tavakoli, A., Culler, D., Levis, P., Shenker, S., Stoica, I.: Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks. In: Proc. SenSys'07 (2007)
6. Lees, J., Crosson, R.: Tomographic inversion for three-dimensional velocity structure at Mount St. Helens using earthquake data. J. Geophys. Res. **94**, 5716–5728 (1989)
7. Li, M., Agrawal, D., Ganesan, D., Venkataramani, A.: Block-switched networks: A new paradigm for wireless transport. In: NSDI'09: Proceedings of the 6th conference on Symposium on Networked Systems Design & Implementation. USENIX Association, Berkeley, CA, USA (2009)
8. Maroti, M., Kusy, B., Simon, G., Ledeczi, A.: The flooding time synchronization protocol. In: Second ACM Conference on Embedded Networked Sensor Systems (2004)
9. McNutt, S.: Seismic monitoring and eruption forecasting of volcanoes: A review of the state of the art and case histories. In: Scarpa, Tilling (eds.) Monitoring and Mitigation of Volcano Hazards, pp. 99–146. Springer-Verlag Berlin Heidelberg (1996)
10. Moran, S., Lees, J., Malone, S.: P wave crustal velocity structure in the greater Mount Rainier area from local earthquake tomography. J. Geophys. Res. **104**(B5), 10,775–10,786 (1999)
11. Murray, T., Endo, E.: A real-time seismic-amplitude measurement system (rsam). In: Ewart, Swanson (eds.) Monitoring Volcanoes: Techniques and Strategies Used by the Staff of the Cascades Volcano Observatory, 1980-1990, vol. 1966, pp. 5–10. USGS Bulletin (1992)
12. Neuberg, J., Luckett, R., Ripepe, M., Braun, T.: Highlights from a seismic broadband array on Stromboli volcano. Geophys. Res. Lett. **21**(9), 749–752 (1994)
13. Paxson, V.: On calibrating measurements of packet transit times. In: Measurement and Modeling of Computer Systems, pp. 11–21 (1998). URL `citeseer.ist.psu.edu/paxson98calibrating.html`
14. Paxson, V.: Strategies for sound internet measurement. In: IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement, pp. 263–271. ACM Press, New York, NY, USA (2004). DOI http://doi.acm.org/10.1145/1028788.1028824
15. Phillips, W., Fehler, M.: Traveltime tomography: A comparison of popular methods. Geophys. **56**, 1649–1649 (1991)
16. Scarpa, R., Tilling, R.: Monitoring and Mitigation of Volcano Hazards. Springer-Verlag, Berlin (1996)
17. Werner-Allen, G., Dawson-Haggerty, S., Welsh, M.: Lance: Optimizing high-resolution signal collection in wireless sensor networks. In: Proc. ACM Conference on Embedded Networked Sensor Systems (Sensys). Raleigh, NC, USA (2008)

18. Werner-Allen, G., Johnson, J., Ruiz, M., Lees, J., Welsh, M.: Monitoring volcanic eruptions with a wireless sensor network. In: Proc. Second European Workshop on Wireless Sensor Networks (EWSN'05) (2005)
19. Werner-Allen, G., Lorincz, K., Johnson, J., Lees, J., Welsh, M.: Fidelity and yield in a volcano monitoring sensor network. In: Proc. 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006). Seattle, WA (2006)
20. Werner-Allen, G., Swieskowski, P., Welsh, M.: MoteLab: A Wireless Sensor Network Testbed. In: Proc. the Fourth International Conference on Information Processing in Sensor Networks (IPSN'05) (2005)