A Wireless Sensor Network For Structural Monitoring*

Ning Xu[†] [‡] Sumit Rangwala[†] Krishna Kant Chintalapudi[†] Deepak Ganesan [§]

Alan Broad [¶] Ramesh Govindan[†] Deborah Estrin [§]

ABSTRACT

Structural monitoring—the collection and analysis of structural response to ambient or forced excitation–is an important application of networked embedded sensing with significant commercial potential. The first generation of sensor networks for structural monitoring are likely to be *data acquisition systems* that collect data at a single node for centralized processing. In this paper, we discuss the design and evaluation of a wireless sensor network system (called Wisden) for structural data acquisition. Wisden incorporates two novel mechanisms, *reliable data transport* using a hybrid of end-to-end and hop-by-hop recovery, and low-overhead *data time-stamping* that does not require global clock synchronization. We also study the applicability of wavelet-based *compression* techniques to overcome the bandwidth limitations imposed by low-power wireless radios. We describe our implementation of these mechanisms on the Mica-2 motes and evaluate the performance of our implementation. We also report experiences from deploying Wisden on a large structure.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Wireless communication; C.3 [**Special-Purpose and Application-Based Systems**]: Embedded Systems

General Terms

Reliability, Design

Keywords

Sensor Network, Structural Health Monitoring, Wisden

1. INTRODUCTION

Structural health monitoring systems seek to detect and localize damage in buildings, bridges, ships, and aircraft. The design of such systems is an active and well-established area of research. When built, such systems would infer the existence and location of damage by measuring structural response to ambient or forced excitation. Wireless sensor networks are a natural candidate for structural health monitoring systems, since they enable dense in-situ sensing and simplify deployment of instrumentation. However, techniques for damage assessment are quite complex, and practical wireless networked structural health monitoring systems are several years away.

Wireless sensor networks do have a more immediate role to play in structural monitoring. Advances in structural engineering depend upon the availability of many detailed data sets that record the response of different structures to ambient vibration (caused, for example, by earthquakes, wind, or passing vehicles) or forced excitation (delivered by large special-purpose shakers). Currently, structural engineers use wired or single-hop wireless *data acquisition systems* to acquire such data sets. These systems consist of a device that collects and stores vibration measurements from a small number of sensors. However, power and wiring constraints imposed by these systems can increase the cost of acquiring these data sets, impose significant setup delays, and limit the number and location of sensors. Wireless sensor networks can help address these issues.

In this paper, we describe the design of Wisden, a *wireless sensor network system for structural-response data acquisition*. Wisden continuously collects structural response data from a multi-hop network of sensor nodes, and displays and stores the data at a base station. Wisden can be thought of as a first-generation wireless structural monitoring system; it incorporates some in-network processing, but later systems will move more processing into the network once the precise structural monitoring applications are better understood. In being essentially a data collection system, Wisden resembles other early sensor networks such as those being deployed for habitat monitoring [10].

While the architecture of Wisden is simple—a base station centrally collecting data—its design is a bit more challenging than that of other sensor networks built till date. Structural response data is generated at *higher data rates* than most sensing applications

*This material is based upon work supported by the National Science Foundation under Grants No. 0121778 (Center for Embedded Networked Systems) and 0325875 (ITR: Structural Health Monitoring Using Local Excitations and Dense Sensing). Any opinions, findings and conclusions or recomendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

[†]Computer Science Department, University of Southern California, {nxu, srangwal, chintala, ramesh}@usc.edu

[‡]Current Affiliation - Center for Embedded Networked Sensing, Los Angeles

[§]Computer Science Department, University of California, Los Angeles {deepak, destrin}@cs.ucla.edu

[¶]Crossbow Technology Inc. abroad@xbow.com

SenSys'04, November 3–5, 2004, Baltimore, Maryland, USA.

(typically, structures are sampled upwards of 100 Hz). Furthermore, this application requires *loss intolerant*[1] data transmission, and *time synchronization* of readings from different sensors. The relatively low radio bandwidths, the high packet loss rates observed in many environments [21], and the resource constraints of existing sensor platforms add significant challenges to the design of Wisden.

In this paper, we discuss the design and implementation of Wisden. Wisden currently uses a *vibration card*, especially designed for structural applications. In addition to describing this card, our description of Wisden focuses on its three novel software components:

Reliable Data Transport Wisden uses existing topology management techniques to construct a routing tree [20], but implements a hybrid error recovery scheme that recovers packet losses both hop-by-hop and end-to-end.

Compression Wisden uses a simple run-length encoding scheme to suppress periods of inactivity in structural response, but we also evaluate the feasibility of wavelet compression techniques to reduce Wisden's data rate requirements and to improve latency.

Data Synchronization Wisden also implements a data synchronization scheme that requires little overhead and avoids the need to synchronize clocks network-wide.

For each of these components, we evaluate its performance through experiments on the motes. We also report experiences from a small deployment of Wisden on a real structure.

Our choice of platform (Mica2 motes, Chipcon radios) is largely dictated by availability of interface hardware (*e.g.,* the vibration card). However, as these platforms evolve to using perhaps ARM-based processors and Zigbee radios, we do not see the need for our proposed mechanisms going away. Rather, such an evolution (particularly to higher-bandwidth radios) will help increase the scale of Wisden deployments. One kind of platform we have explicitly not considered are more powerful processors equipped with 802.11 radios. While there has been some work on using high-powered radios as cable replacements in wired instrumentation systems [9], these platforms are significantly more energy-intensive, and have less software support for multi-hopping (which can increase deployment flexibility).

2. BACKGROUND AND MOTIVATION

In this section, we first discuss the requirements of structural monitoring and describe devices used to measure structural response. We then discuss structural data acquisition systems, their capabilities and their shortcomings. This sets the stage for the design of Wisden, which we discuss in the next section.

2.1 Sensing Structural Response

Structural engineers use different kinds of sensors to monitor structures: displacement sensors, strain gauges, and accelerometers, to name a few. While Wisden can be used with displacement sensors and strain gauges, we focus in this paper on an accelerometer-based system. Accelerometers measure, as the name suggests, accelerations of the surface they are mounted on. Accelerations are translated into changes in capacitance or in other electrical properties. These analog signals are then sampled at a specified frequency.

[1]We should note that Wisden's silence suppression technique is lossy, but the system attempts to deliver useful structural vibration data reliably.

From a structural engineering standpoint, accelerometers are characterized by several performance parameters: sensitivity, which denotes the smallest measurable acceleration and is expressed in g's (gravitational acceleration); dynamic range, which denotes the range of accelerations that the device is capable of measuring and is also expressed in g's; and noise, which is measured either as an RMS value, or is expressed as a function of the frequency of vibration.

From a software designer's perspective, the output of an accelerometer is a time series of sensor readings with a specified resolution and a specified sampling rate. Strictly speaking, these are parameters associated with the analog-to-digital circuitry attached to an accelerometer, but they nevertheless constrain the performance of the accelerometer. The resolution constrains the sensitivity of an accelerometer; a 10-bit accelerometer whose dynamic range is $1g$ cannot have a sensitivity less than $1\text{m}g$. The sampling rate, on the other hand, governs the frequencies that can by measured by the accelerometers.

Accelerometers are used for a wide variety of applications. For monitoring large structures, though, it is generally considered sufficient to have a dynamic range of 1-2 g's, a sensitivity in the μg range and low noise characteristics. This translates to a sampling resolution of at least 16 bits per sample. Finally, since many structural engineering methods monitor the frequency response of structures (which are usually focused in the tens of Hz), a sampling rate of 100 Hz is considered to be a minimum requirement.

Thus, from our perspective, an accelerometer might be modeled as a device that generates about 100 2-byte samples a *second*. More generally, a single sensor node might be attached to an accelerometer capable of measuring accelerations along three axes. Such triaxial accelerometers are capable of generating about 600 bytes a second, which is a significant fraction of the bandwidth of current and future low-power wireless radios such as the Chipcon CC1000, and the Zigbee (IEEE 802.15.4) radios.

2.2 Structural Data Acquisition Systems

Accelerometers (or displacement sensors and strain gauges, for that matter) collect structural response from a single location on a large structure. Structural engineers would like to collect data from tens or hundreds of locations. A long-term goal of such an instrumentation infrastructure is an on-line system for damage detection and localization. In such a system, structural response from different locations can be used to parametrize a model of the structure; when damage occurs, the parameters of this model change, allowing the system to infer the existence (and possibly location) of damage. Practical damage detection and localization systems are several years away. As an aside, there is no inherent reason for such systems to be centralized; wireless sensor networks employing decentralized detection and localization algorithms are plausible, but are beyond the scope of this paper.

In order to develop methods for detecting and locating damage, structural engineers rely on extensive data-sets of structural response. These data-sets can help validate, benchmark, and provide intuition for such methods. Currently, these data sets are collected by expensive wired (or one hop wireless) and powered *data acquisition systems*. These systems typically consist of a single device that supports a fixed number of *channels*. Each channel is connected to one sensor. Data acquisition systems include sophisticated signal conditioning, processing and analysis functions. A simpler, and cheaper, variant of a data acquisition system is a *data logger*—it lacks some of the analysis capabilities of a data acquisition system, and merely provides storage and high-bandwidth transmission capabilities for the collected data.

Data acquisition systems collect structural response either to ambient vibrations, or forced vibrations. Ambient vibrations can be caused by earthquakes, or passing vehicles. In large structures such as bridges and buildings, wind can also evoke structural response. Obviously, systems that collect structural response to ambient vibrations are generally long-running, since the occurrence of significant ambients may be unpredictable. For this reason, especially from structures under test, engineers collect structural response to *forced* vibrations. Occasionally, these are delivered by large *shakers*: mechanical devices with large moving parts attached to the structure, whose motion vibrates the structure at different frequencies.

In either scenario, setting up a data acquisition system is an expensive proposition and a cumbersome endeavor. Data acquisition systems are expensive, which limits the number of points on the structure that can be instrumented. Furthermore, installing the cabling for the sensors and the power for the data acquisition system itself is a logistical challenge. Anecdotally, engineers reckon preparing a structure for data collection can take 2-3 weeks.

In this paper, we consider whether wireless sensor networks can potentially replace structural data acquisition systems. Sensor devices offer the freedom from cabling and associated placement constraints. This, together with multi-hop routing, allows a very flexible instrumentation infrastructure. On the other hand, several constraints of sensor networks make it difficult to design a wireless structural data acquisition system. Most important of these is energy: not only is wireless communication energy-intensive, but accelerometers themselves are not low-power as we discuss later. Other challenges include limited radio bandwidths, high packet loss in wireless environments, and lack of time synchronization. We return to these challenges later.

Given our discussions above, a wireless long-lived (for several weeks) structural data acquisition system for measuring response to ambient vibrations will require careful systems engineering. We do not undertake this endeavor in this paper. Rather, we focus on the design of a structural data acquisition system that can be deployed for a short-term (a few hours to a day), such that it is possible to provision adequate battery power. Usually, such a system will be deployed in situations where forced vibrations are used to excite a structure. In these situations, a wireless system has two advantages: rapid deployment and flexibility. Both these advantages are evident in the following actual episode. A transportation agency is ready to declare a newly built bridge open, but allows a team of structural engineers one or two days to measure structural properties by, for example, driving a large truck through the bridge. While wiring such a structure might take hours to days, a wireless network can be deployed in tens of minutes. Often, the challenge in such scenarios is not knowing *where* to instrument the structure. This is usually because the structural characteristics may not be precisely known. A wireless data acquisition system allows the engineers to iterate on sensor placement to determine appropriate locations.

3. Wisden: BACKGROUND AND DESIGN OVERVIEW

In this section, we discuss the design of Wisden, our structural data acquisition system. We first describe the hardware that we use, then present the abstraction that the system provides to the user, and finally give a brief overview of how the system works. In the subsequent sections, we describe the system internals in detail.

3.1 Hardware

Wisden uses mostly off-the-shelf hardware. Specifically, our sensor nodes are the Mica-2s. The Mica-2 represented a convenient low-power platform which already had a few software components that we could re-use for our purpose. Although careful power management and recording ambients was not a goal of our project, moving in that direction is now easier since we have started with the Mica-2. However, the memory constraints of the Mica-2 made it a slightly more difficult platform choice, since our application is memory intensive.

Existing sensor platforms do not, however, have hardware support for high quality vibration sensing. So, Wisden uses a 16-bit *vibration card* designed specifically for high-quality vibration sensing. The card was originally designed for high-frequency (up to 20ksps), sampling at 16 bits per sample. It consists of 4 separate analog input channels. Each channel has sensor excitation (5V or 18 V constant current), gain, attenuation and a programmable anti-aliasing filter. The analog channels are interfaced to a 16-bit analog-to-digital converter. The ADC is controlled by an on-board microprocessor, and exhibit a sensitivity of 100mV/g.[2] In turn, the microprocessor can be commanded by an attached Mica2 mote (which runs the Wisden software) to set the sampling rate, read the output data, and stores the samples into an external 64K byte SRAM.

The vibration card is designed for low power operation. Two separate, controllable, power supplies allow the card to power up different circuits. One of the circuits supplies power to the on-board microprocessor and is enabled by a control line from the Mica2. Once the microprocessor is powered, commands from the mica2 then enable analog power to the sensors and signal conditioning circuits. Full power is only used during data acquisition; during this time, the current draw is about 100 mA. After data is stored in the on-board SRAM the analog power can be disabled while data is being retrieved. Data can be retained in the SRAM at a very low sleep current ($< 50\ \mu$A) until needed.

The vibration card was not specifically designed for our application: lower frequency (100 Hz) *continuous* sampling. As such, it draws rather more power than one would expect from a board specifically designed for our application. We believe we can reduce the current draw by customizing the board to our application using various tricks: removing the on-board microprocessor and the SRAM, and multiplexing the signal conditioning circuitry across the different channels. We estimate that the current draw in such a board will be in the 40 mA range.

Finally, we modified the on-board microprocessor's software to permit continuous sampling. Our modified software allows periodic, sample-by-sample, data acquisition and does not use the on-board SRAM on the vibration card. Getting our new firmware working proved to be a significant software engineering challenge because there was a pin conflict between the vibration card and the EEPROM which Wisden uses to store vibration samples. We finally solved this using some tricky low-level handshaking to arbitrate the use of the conflicting pin.

3.2 Wisden Abstraction

The goal of Wisden is to provide the abstraction of a data acquisition system (in its current form, Wisden provides the functionality of a data logger as it lacks the on-line data processing capabilities present in data acquisition systems). Traditional data loggers support a fixed number of *channels*. By contrast, Wisden can support a flexible number of channels by trading off acquisition latency or if the measured vibration activity is intermittent (see below). In all other respects, Wisden seeks to emulate data loggers as closely as

[2]The accelerometer used with the vibration card provided a low noise of 300μgrms.

possible. Of course, the implementation of Wisden is quite different from that of traditional data loggers, as we discuss below.

A data logger or acquisition system abstraction implies that samples are centrally collected in near real-time. An alternative approach would have been to design a system where the data at each sensor is stored at the sensor node and retrieved later manually. The storage limitations on current platforms limit the amount of vibration data one could collect. More fundamentally, however, such a system would be cumbersome to use. It would also not allow structural engineers to iteratively decide where to place the accelerometers, a crucial requirement for data collection.

3.3 Wisden Overview

Although the abstraction presented by Wisden is simple, its design and implementation is rather challenging. A typical Wisden deployment will consist of several tens of nodes placed at different locations on a large structure. Each node has an attached accelerometer that is capable of sensing up to three channels of vibration data, with a configurable sampling rate. A *base station* provides the functionality equivalent to a data logger or acquisition unit—the ability to store samples and to provide near real-time display of samples. Nodes self-configure to form a tree topology, then send their vibration data to the base station, potentially over multiple-hops.

Implicit in the data acquisition system abstraction that Wisden provides are two essential design requirements: that the vibration samples be delivered *reliably* to the base station, and that samples be *time-synchronized*. Further complicating the design of the system is the fact that the data rates from a single sensor node can be a significant fraction of the radio bandwidth available on current sensor platforms. For example, a tri-axial accelerometer generating 16-bit samples at 100 Hz requires 4.8 Kbps. The Chipcon radio nominally provides 19.9 Kbps after accounting for coding overhead, but achievable radio data rates are closer to 10 Kbps.

Clearly, the bandwidth limitation implies that reliably delivering *every* sample from tens of nodes is infeasible. Fortunately, structural engineers are content to acquire vibration data corresponding to interesting *events* – relatively large motions caused by earthquakes, high wind, or large vehicles.[3] Accordingly, nodes in Wisden locally *compress* the data before transmitting it to the base station. Of course, this approach serves to reduce energy usage as well, and represent the kind of in-network processing that sensor networks are predicated on. Our current implementation of Wisden incorporates a simple run-length encoding scheme, but we also implement and evaluate more sophisticated wavelet compression schemes. We have not yet integrated the latter into Wisden given memory limitations.

Once the nodes generate compressed data, they transmit data to the base station. Data transmitted by a node is rate-limited, and the rate limits are currently manually configured (we discuss this later in detail). Wisden nodes implement a *hybrid hop-by-hop and end-to-end* error recovery scheme to enable reliable transmission of data to the base station. In this scheme, vibration samples are recovered, to the extent possible, in a hop-by-hop fashion. Wisden's error recovery protocol aggressively uses overhearing and piggybacking techniques in order to detect and repair packet loss. (We can do this because, at least in its current design, Wisden does not put nodes or radios into low-power states.) Given high packet losses on links, this approach can reduce the overhead of error recovery. However, topology changes and high packet loss rates necessitate a fallback end-to-end recovery scheme that ensures reliable end-to-end delivery.

Finally, Wisden also implements a *data time-stamping* scheme that is qualitatively different from the time synchronization schemes discussed in the literature [3, 4]. In this scheme, as a packet is transmitted to the base station hop-by-hop, it acquires enough timing information for the base station to determine when (by its local clock), the corresponding samples were generated by the sending node. In this fashion, samples corresponding to the same event, but generated by two different nodes, can be synchronized *at the base station*[4].

The rest of the paper discussed these three main components of Wisden: reliable data transport, compression and data time-stamping. In each section, we discuss the design of each component, then evaluate it using our implementation. We follow this with a section that presents preliminary measurement data from an actual structure.

4. RELIABLE DATA TRANSPORT

The first challenge in Wisden is to reliably transmit data from each sensor to the base station. Techniques for reliable transport in networking are well-studied in the networking literature and Wisden leverages many of these techniques. In particular, Wisden uses *both* hop-by-hop and end-to-end recovery; the former is a necessary performance optimization in wireless networks where link losses of up to 30% are not uncommon [21].

In our current implementation of Wisden, nodes first self-organize into a tree topology. Each node stores the generated vibration data into its EEPROM (after run-length encoding compression, described in the next section), and then transmits it to the base station. An aspect closely related to reliable transport is sending rate adaptation. In our current implementation of Wisden, node transmission is rate-limited to a configured value. In this section, we describe the topology self-configuration and data transport components of Wisden, and evaluate its performance.

4.1 Related Work

Reliable, congestion-adaptive data transport for wireless sensor networks is an ongoing area of research. Recently several reliability protocols have been proposed. RMST [13] (Reliable Multi-Segment Transport) is a transport layer protocol that is designed to add reliable service on top of Directed Diffusion. RMST is a NACK-based protocol, receiver detects loss and loss is repaired hop-by-hop. In this respect, it most closely resembles Wisden's mechanisms, but because it is integrated with Diffusion, designed for larger platforms, and optimized for recovering losses of image fragments, it was unsuitable for our application. PSFQ [17] (Pump Slowly, Fetch Quickly) is a hop-by-hop reliable transport protocol designed for sensor network reprogramming. In this application, the direction of data transfer is from a base station to all the nodes in the network. In Wisden, data transfer is in the opposite direction.

There has also been some recent work on congestion control for wireless sensor networks. Two examples of such work are CODA [18] and ESRT [12]. Both these pieces of work are less concerned with reliable data transfer from sources to sinks. Rather, they are designed for applications in which an event may be detected by correlated sources, and the system needs to avoid a degra-

[3] By contrast, seismologists, who sometimes monitor structural responses to micro-tremors, usually require even low-levels of vibration to be recorded. Another interesting difference is that seismologists are interested in 24-bit data, so the current instantiation of Wisden is insufficient for their needs.

[4] Of course, this technique generalizes easily to synchronizing samples at intermediate network nodes (*e.g.*, at a common ancestor in the sink tree), but we have not implemented this in Wisden.

dation in the fidelity of reported data when congestion occurs. Also in this vein is early work by Woo *et al.* [19] on rate adaptation in sensor networks. That work considers periodic sources sending data to a sink, and the goal is to adapt the sending rate to the perceived congestion in the network. Reliable transport is not a goal of that piece of work.

There has been much work on reliable transport in wired networks and wireless mobile networks. For brevity, we do not cover that literature here.

4.2 Topology Self-Configuration

In Wisden, nodes self-organize themselves into a routing tree rooted at the base station. This problem has been extensively studied by Woo *et al.* [20] who showed that it is important to use only "good" wireless links in the tree topology in order to get good performance. In their approach, nodes select parents based on packet loss performance to potential parents. This packet loss performance can be measured both passively (using actual data transmissions) and actively (using probes sent by nodes).

Wisden actually leverages the software developed for [20]. It uses the prototype called BLAST [20], which supports both tree construction and packet delivery.[5] Specifically, BLAST has two separable components: one that performs parent selection, and another that exports data transmission and reception interfaces. Wisden uses only the parent selection component and implements its reliable transport on top of it. An important consequence of this is that Wisden uses active probes and not data traffic in order to estimate link quality. While this does not affect the correctness of Wisden it does add additional overhead. Reducing this overhead will be a focus in the next version of Wisden. We did not need to modify the BLAST parent selection module at all, save for setting application-specific parameters that control the probing rate.

4.3 Implementation of Reliable Data Transport

Wisden implements a hop-by-hop NACK-based reliability scheme. Each source stores generated vibration data in its EEPROM, then transmits the data to its parent. Parents keep track of sequence numbers of packets that they receive, on a per source basis. A gap in the sequence number of sent packets indicates packet loss. Each node maintains a list of missing packets. When a loss is detected, a tuple containing a source ID and sequence number of the lost packet is inserted into this list. Entries in the "missing packets" list are piggybacked in outgoing transmissions, and children infer losses by overhearing this transmission. Nodes keep a small cache of recently transmitted packets, from which a child can repair losses reported by its parent.

Lost packets are often recovered hop-by-hop, however, two factors necessitate *end-to-end* recovery. First, heavy packet losses can lead to large missing packet lists that might exceed the memory of the motes. We have observed this in our experiments. More fundamentally, a topology change could cause loss of missing packet list information. For example, when a node selects a new parent, it will no longer respond to repair requests for missing packets from its previous parent.

Our *end-to-end* recovery scheme is essentially implemented in much the same way as our hop-by-hop scheme. It leverages the fact that the base station has significantly more memory and can keep track of all missing packets. The base station attempts hop-by-hop recovery of a missing packet. When one of its children notices that it has seen a packet from the corresponding source, but does not have a cached copy of that packet, it adds that recovery request to its missing packets list. This request is propagated downward in this manner (using the same mechanisms described for hop-by-hop recovery) until it reaches the source. Since the source maintains generated packets in its EEPROM, it can repair the missing packet.

During testing, we noticed an interesting livelock in our implementation of this recovery, caused by memory constraints on the Mica-2. When some children (particularly those who are listed in the parent's missing packets list) of a parent migrate to a different parent (*e.g.*, when wireless link quality changes), the parent's missing packets list is never cleared, so that new entries cannot be added and the system never makes progress. To avoid this, we use a simple watchdog timer that resets the entire missing packets list at a node. This timer is set whenever a missing packets list becomes full, and is cleared whenever any entry is removed from the list.

Finally, if there is no radio transmission for a certain period of time, a dummy packet with sequence number N+1 is sent out, where N is sequence number of the last outgoing packet. This dummy packet serves 3 purposes: it detects the loss of the last packet in a stream, it triggers recovery, and maintains state about parent-child relationships at the node's children.

Most of the reliability mechanisms are implemented on the Mica-2. Figure 1 shows the block diagram of our reliability implementation in TinyOS. However, a crucial component is implemented at the base station (as alluded to above). Wisden relies on a PC-class machine as a base station, connected to a mote through the serial interface. The PC implements end-to-end recovery, as well as as data decompression and time-stamping (described in subsequent sections). Functionality on the PC is implemented in Java.

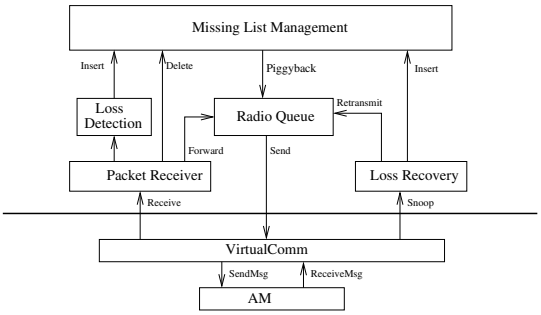

Figure 1: Block Diagram of Reliability Implementation

Wisden currently does not include two important pieces of functionality. Adding end-to-end cumulative acknowledgements would allow sources to purge the vibration data from EEPROM, a necessary pre-requisite for a deployable system. This is easy to do. A bigger limitation of Wisden is that it requires manual configuration of a sending rate. This rate is a function of the achievable radio bandwidth and the number of nodes in the network. In our tests, the achievable radio bandwidth R is measured empirically by sending packets back-to-back and determining a "safe" rate (one at which significant packet losses do not occur). Then, each Wisden node is configured to so that vibration data that it originates is rate limited to $\frac{R}{N}$ where N is the number of nodes. Rate-adaptation is related to congestion control, and is a research topic in and of itself. We have left this to future work.

4.4 Experimental Evaluation

How well does our scheme work? We deployed 25 mica2 motes on three floors of a medium-sized office building. Fifteen of those motes were programmed to generate *artificial* traffic, not vibrations

[5] BLAST is a precursor to the MintRouting TinyOS component. For logistical reasons, Wisden uses BLAST, but we intend to migrate to MintRouting soon.

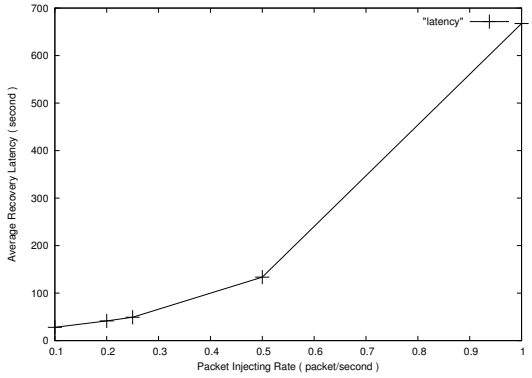

Figure 2: Average Recovery Latency

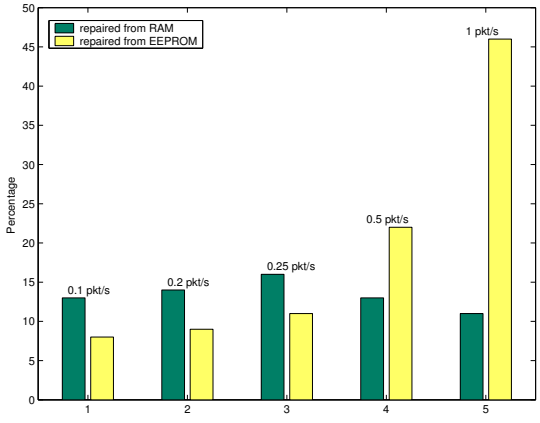

Figure 3: End-to-end vs. hop-by-hop, analysis based on a mote 3 hops away from the sink.

recorded from an accelerometer. (In this part of the paper, we are interested in evaluating the reliability scheme alone. In a later section, we describe a live experiment on a real structure). The remaining 10 motes only forwarded traffic, and were placed to improve the density of the deployment as well as to ensure that we got a multihop topology. Having these ten nodes increases the likelihood of selecting different parents at different times, and triggers the end-to-end recovery[6].

We performed several experimental runs. In each run of the experiment, the 15 nodes programmed to generate traffic did so at one of the following rates: 0.1 packet/sec, 0.2 packet/sec, 0.25 packet/sec, 0.5 packet/sec and 1 packet/sec. Each packet was 80 bytes long and carried 18 samples. For context, a 1 packet/sec rate can enable the network to support readings from a structure that vibrates about 18% of the time (assuming no packet losses). In each run, every node sent out 200 packets, resulting in a total of 3000 packets per run.

In all our experiments, we achieved 100% reliability. However, if we set the sending rate to 2 packet/sec per node, our network essentially collapses and very few of the packets are received. This collapse is hinted at in Figure 2, which plots the average end-to-end latency experienced for each experiment. Notice how this latency increases dramatically for the highest sending rate; in this regime, packet recovery plays a dominant part in the latency. This has important implications for our time-stamping scheme, as discussed in Section 6. These results also point out the need for an intelligent rate-adaptation scheme that tries to keep the nodes close to the knee of the curve. Finally, Figure 3 shows how many packets are transmitted from the EEPROM (this corresponds to sources generating retransmissions), vs. those transmitted from RAM (this corresponds to intermediate nodes retransmitting packets). Notice, how, beyond 0.25 packets/sec, the system moves into a regime where end-to-end recovery dominates (far more packets recovered from the EEPROM). This, of course, explains the increased latency that we see in Figure 2.

[6]We should mention that these 10 nodes also improve connectivity. With a smaller deployment, the packet losses can be higher, and this impacts recovery latency, as we discuss a bit later.

5. COMPRESSION

The second crucial aspect of Wisden is *data compression*. While most prior research ([7]) has focused on data aggregation in order to increase network lifetime, our primary motivation for considering compression is to scale Wisden to many nodes. In Section 3.3, we showed how the data rate requirements for structural monitoring can be a significant fraction of radio bandwidth. Another way to describe this limitation is to consider the *latency* of data acquisition; Even if each of 20 nodes generated only 10 minutes worth of 3-channel vibration data, it would take almost an hour transmit the data to the base station assuming a nominal radio bandwidth of 2 KBps.

Higher bandwidth 802.11 radios represent a possible solution to this problem. However, platforms employing such radios typically consume an order of magnitude more power. As such, each node would require significantly large batteries in order to run unattended for up to a day. While this is not a fundamental challenge, it does present significant practical difficulties. An important consideration in large structures is *mounting* the instrumentation safely (to avoid disconnections or damage to the instrumentation). Lighter, smaller platforms like the motes and their associated sensor boards and batteries can be quickly taped onto structures. Larger, more powerful hardware platforms with heavy batteries might need careful drilling and mounting, together with cabling that allows flexible placement.

For this reason, we consider a second alternative, data compression. One possible approach to data compression is *event detection*; only transmitting samples that exceed a certain threshold, and utilizing the fact that structures will experience relatively few of these. Our current implementation of Wisden uses this approach. A more general strategy is to use *progressive storage and transmission* that stores vibration data locally and transmits a lossy version (using wavelet compression) of the data to the base station. Such an approach enables low-latency but lossy data acquisition. The stored data allows detailed views of the vibration data to be retrieved on demand. This technique will be useful in platforms that have significant local storage, a trend that is likely given the falling prices of flash memory. In the rest of this section, we describe these approaches and evaluate them.

5.1 Related Work

Event detection and data compression are very elaborately studied research areas. Our major contribution is in applying these ideas to a new domain (wireless sensor networks), understanding its applicability to new datasets (wireless structural vibrations) and its implementation on highly resource-constrained devices (motes). We briefly describe some key prior research that relate to our work.

Dimensions [5] is an in-network sensor data storage system that maintains summaries at different resolutions constructed using wavelet compression. While our work bears similarities to Dimensions in the use of wavelet compression, we differ in system goals and the choice of implementation platform. Our investigation of wavelet

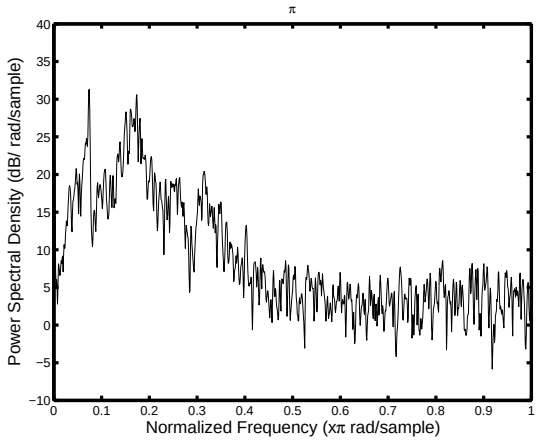

Figure 4: Periodogram of the Power Spectral Density estimate of the structural vibration event. Energy is concentrated in the low-frequency bands, making the use of wavelet compression ideal.

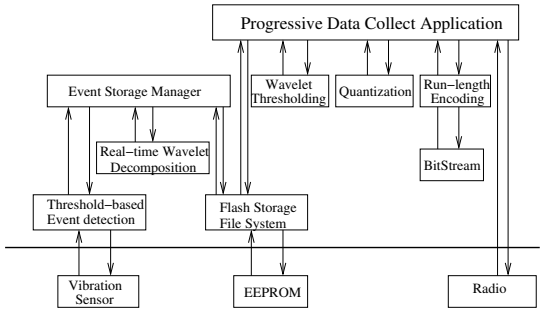

Figure 5: Component Diagram of Mote implementation

techniques is designed with latency in mind rather than as technique for distributed storage, hence our focus is on using wavelet coding for progressive transmission. In addition, our implementation is on the mote platform rather than for the IPAQs.

Lynch *et al.* [9] propose the use of wavelet compression for loss-less data transmission of structural monitoring data. While our reasons for using wavelet compression are similar, our system is designed for lossy transmission rather than lossless.

Progressive transmission has been used extensively in the internet context, especially for transmission of large images over bandwidth-limited or latency-constrained links. Our work uses ideas from progressive transmission schemes such as JPEG2000 [14], however, our design and implementation targets a significantly more resource-constrained platform.

5.2 Event Detection

The simplest approach to compression of vibration data is *event detection*. This approach is based on the observation that, if samples within a small window have a low value *and* are comparable in value, the structure is quiescent. Such quiescent periods are compressed using run-length encoding; samples in non-quiescent periods are transmitted without compression.[7]

Event detection suppresses data transmission when events do not occur. Thus, the overall data rate required to transmit the samples is a function of the duty-cycle of the vibrations, and directly affects the Wisden's scaling. For example, if continuous vibration data from two nodes matches the radio bandwidth, then vibration samples can be collected from a network of 20 nodes when the structure shakes only 10% of the time.

This approach has two limitations. The first is that the number of instrumentation locations is constrained by the rate at which a structure is expected to vibrate. For forced vibrations, this might constrain the design of the experiment. Second, and more important, this approach does not reduce the user-perceived *latency* of data acquisition. Due to the global nature of vibration events, vibration data is usually generated from all node simultaneously. In the example discussed at the beginning of this section, even if the 20 nodes generate 1 minute of vibration data (a duty cycling of 10%), it would take 6 minutes to transmit all the data to the base station.

5.3 Progressive Storage and Transmission

To address the latency of data acquisition, we have designed and implemented a progressive storage and transmission strategy on the motes. This approach uses local storage on the motes as a in-network cache for raw data and transmits low-resolution compressed summaries of data in near-real time. The user can visualize this summarized event data and the raw data can be collected from the distributed caches when required. This on-demand data collection has to occur within the time window before which data in the cache is replaced by newly generated samples. As we show below, such an approach can compress vibration data by a factor of 20; when coupled with event detection, it can reduce the acquisition latency to less than a minute in many cases.

Our progressive transmission strategy for vibration data uses wavelet compression. The applicability of wavelet techniques follows from characteristics of large structures, whose frequency response is usually focused in the low-frequency components [16]. Figure 4, which shows the power spectral density of a vibration signal, illustrates this clearly.

5.3.1 Wavelet Codec Internals

We use an optimized implementation for the motes due to memory and computation constraints. For this reason, many design choices that we make are simpler than other progressive codecs such as JPEG2000. The component diagram of our implementation is shown in Figure 5. We now describe the individual system components in more detail.

Integer-Integer Wavelet decomposition: Our implementation uses the bi-orthogonal Cohen-Daubechies-Feauveau (2,2) (CDF(2,2)) integer wavelet lifting transform that relies solely on integer addition and bit shifting operations. Wavelet lifting involves two steps: (a) a prediction step when the odd values of the time-series are predicted from the even values, and (b) an update step when the even values are updated to capture the error in the prediction step. The predict and update operations for the CDF(2,2) lifting transform are:

$$d_i \leftarrow d_i - \frac{1}{2}(s_i + s_{i+1})$$

$$s_i \leftarrow s_i - \frac{1}{4}(-d_{i-1} - d_i)$$

The choice of the above lifting transform over other kernels was based on two factors: computation overhead and compression performance. Using longer wavelet filters involves more computation overhead but does not provide significant compression improvement over the chosen filter, at least for the building vibration dataset that we studied ([8]).

[7] In practice, these samples can be compressed using some loss-less compression techniques, an approach we have not yet investigated, but which we do not expect to provide significant gains.

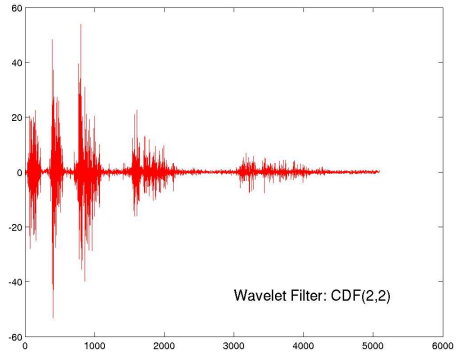

Figure 6: Wavelet Decomposition

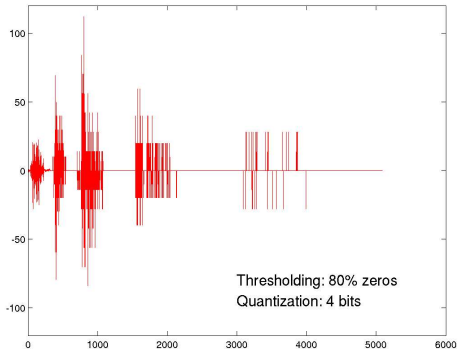

Figure 7: Quantization and Thresholding

While the lifting transform itself is very efficient, normalization of coefficients at various subbands involves floating point operations. The normalization coefficients for the CDF(2,2) transform are:

$$n_H = \sqrt{2} \qquad (1)$$
$$n_L = \frac{1}{\sqrt{2}}$$

where n_H is the higher frequency subband and n_L is the lower frequency subband. We perform the normalization operations during the wavelet thresholding step rather than during wavelet decomposition to be more computationally efficient.

The wavelet codec operates on buffers of length 2^n, where n is a positive integer. To avoid blocking artifacts at the buffer boundaries, we pad each buffer with a few samples at either end.

Quantization: Quantization involves representing a range of values in a signal by a single value. This reduces the number of symbols that are required to represent a signal, and hence makes the signal more compressible. We implemented a simple uniform quantizer that can be used to reduce the resolution of data depending on the range of the signal and the number of bits allocated to each sample. Figure 7 shows the quantized and thresholded version of the signal in Figure 6.

Signal Thresholding: Thresholding is a technique used to modify the wavelet decomposed signal such that the resulting signal contains long sequences of zeros that can be efficiently compressed by an entropy coding scheme. We use a hard thresholding scheme in which if the absolute value of any wavelet falls below the threshold, it is set to zero (shown in Figure 7). We maintain a probability

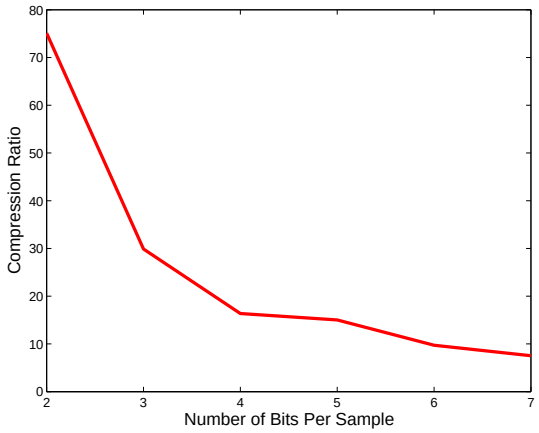

Figure 8: Compression Ratios

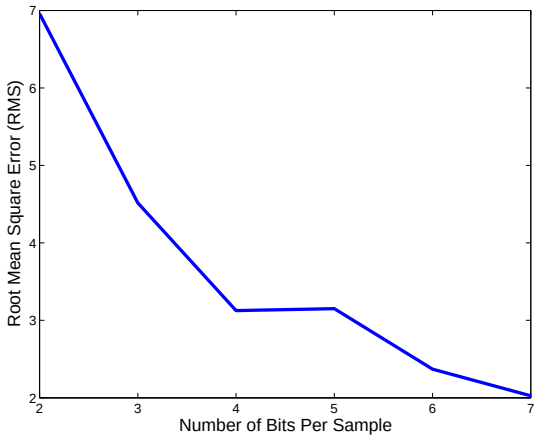

Figure 9: Root Mean Square Error

density function (PDF) of the signal to facilitate the selection of an appropriate threshold. The user specifies what percentage of the signal need to be zeros in the lossy version, and the PDF can be used to determine the appropriate threshold.

The thresholds for different subbands are normalized using the coefficients shown in Equation 1. This operation needs to be done only once, hence, it reduces the computation requirements of normalizing the signal.

Run-length encoding: Wavelet decomposition, quantization and thresholding process the signal to make it more amenable for compression by an entropy coding scheme, but no compression has yet occurred. An entropy coding scheme is typically designed such that the symbols that occur most frequently use the least amount of bits. Run length coding is the simplest of such schemes that exploits *runs* of a particular value in the signal. The thresholded and quantized signal in Figure 7 shows many runs of zeros that can be exploited by such an encoding scheme.

BitStream: The run-length encoded signal is a series of symbols of different lengths depending on the number of bits used in quantization and the lengths of the special symbols used in the run length encoding process. A bitstream module is used to pack these variable length symbols into the data segment of a TinyOS message packet. Each time the data segment of a packet is filled up, it can be scheduled for transmission to the base station using our reliable transport mechanism (Section 4).

5.3.2 Operation Description

The progressive transmission operation involves three steps: event detection, local data storage, and progressive coding. The event detection scheme that we described in Section 5.2 runs continually, and triggers when an event is detected. The event signal then undergoes wavelet decomposition, and the decomposed signal is written to the persistent external flash memory on the mote. Until this point, no compression has occurred, hence, the lossless event data is available on flash.

A separate periodic task reads the flash memory and compresses the signal using the five step process described in Section 5.3.1, after which it transmits the bitstream to the base-station. The choice of signal threshold and number of quantization bins is assumed to be determined by a priori analyis of training data to obtain maximum compression benefit within the specified error bounds.

A user at the base-station can analyze the low-resolution signal in real-time and request either the raw data or additional detail in the signal. Since the raw data is stored on flash, the difference between the previously transmitted resolution and the requested resolution is coded by the steps described in Section 5.3.1 and transmitted to the base-station. This progressive transmission procedure should be completed before the data on flash is overwritten by future events.

This implementation is currently not integrated into the rest of our system due to the need for significant memory optimization. Our evaluation of these ideas is based on a standalone implementation; we intend to integrate this implementation into Wisden within the next few months.

	Computation Time	Memory Utilization
Wavelet Decomposition	6.44ms	288bytes
Uniform Quantizer	0.32ms	7bytes
Run-length Encoder	6.30ms	20bytes

Table 1: Performance of 128-sample 4-level transform

5.3.3 Performance Evaluation

We evaluate the performance of our system on two fronts: (a) the computation and memory overhead of doing the wavelet compression in real-time on a mote, and (b) the compression gain by using our scheme, which translates to the latency of data acquisition. We used data from shaker table tests at CUREE-Kajima [8][8].

Table 1 shows the computation and memory requirements of the core components of our compression system. The computation time is low, and enables us to perform the entire operation in real-time. We were able to perform sensor data sampling, 128 sample CDF(2,2) wavelet lifting transform as well as writing the decomposed buffer to the EEPROM for sampling rates upto 250Hz. As can be seen in Table 1, the memory requirements are low as well, making it easier to integrate with other components of the system.

The compression and error results from using such a scheme are shown in Figure 8 and Figure 9 respectively. Both graphs use a threshold that sets 80% of the decomposed signal to zero. The trends of both graphs are as expected; as the number of quantization bits increases, both the compression ratio and the RMS error reduce. One sweet spot that emerges from these two graphs is a 4-bit quantization scheme. This choice gives us about 20-fold reduction in data size with very low RMS error of 3.1. The peak-to-peak signal to noise ratio (PSNR) for the above choice of parameters is 30dB.

These results are very promising are indicate that such an approach can be used to allow near real-time structural data acquisition from tens of sensors. We expect to more directly validate the latency benefits when we integrate our implementation into Wisden.

[8]While this represents only vibration data from a single structure, we don't expect the results to be different for other large structures for the reason discussed above. The vibration data collected by Wisden and reported in Section 7 arrived too late for analyzing here.

6. TIMESTAMPING THE DATA

Most data loggers provide the ability to time-stamp samples collected by different sensors. Samples need to be accurately time-stamped in order to correlate readings from different sensors. Even if the data sets are used only for frequency analysis, it is necessary to timestamp the samples in order to distinguish responses due to different events.

What level of time-stamping accuracy does Wisden need? To be consistent with the abstraction we attempt to provide, we have conservatively assumed that individual *samples* from different sensors need to be time-stamped *consistently*. By this we mean that if two sensors generate a sample at global time T in response to the same event, Wisden must assign that sample a timestamp T at the base station, regardless of what the local clock readings are.

Clearly, if GPS devices were available at every node, this would not be a problem, but the unsuitability of GPS for wireless sensor networks has been well documented [3]. Sensor network time synchronization schemes [3, 4] are sufficient to solve this problem.

Wisden uses a light-weight approach in that it focuses on time-stamping the data consistently *at the base station*, rather than synchronizing clocks network-wide. This approach requires the addition of a small number of bytes to each packet, but otherwise incurs no messaging costs. In this section, we describe Wisden's approach.

6.1 Related Work

Most relevant to our work is the class of time synchronization schemes that have been recently studied in the context of wireless sensor networks. Most of the proposed schemes address pairwise clock synchronization between nodes, and network-wide synchronization to a reference clock.

RBS [3] is a receiver-receiver scheme where receivers can synchronize with each other using a reference broadcast from a sender. By contrast, sender-receiver synchronization schemes like TPSN [4] use pairwise message exchanges, much like NTP [11] to calculate clock offsets. Both these schemes measure, and compensate for, clock drift. These approaches, as well as that discussed by Rabaey *et al.* [15], propose to achieve network-wide synchronization by recursively synchronizing clock pairwise.

Our approach is much closer in spirit to the post-facto synchronization proposed by Elson *et al.* [2]. In this scheme, nodes' clocks are kept unsynchronized. Events are synchronized *post-facto* using simple beaconing messages.

6.2 Basic Design

In Wisden, each node calculates the amount of time spent by a sample at that particular node using its local clock. This amount is added to an *residence time* field attached to a packet (for simplicity, Wisden associates offsets with the first sample in a packet), as the packet leaves the node. Thus, the delay from the time of generation of the sample to the time it is received by the base station (or any node) is stored in the packet as the sample travels through different nodes in the network. This is the time the packet resides in the network. The base station (or any node) can thus calculate the time of generation of the sample by subtracting the residence time

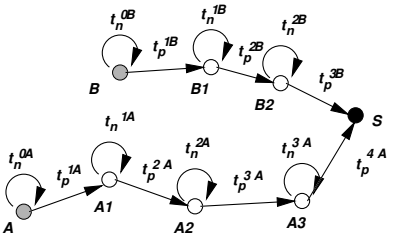

Figure 10: Time synchronization example.

from its local time. If the base station is GPS synchronized, this approach gives a good approximation. If the residence time field is updated as close to the radio and the accelerometers as possible, then, assuming packet propagation times are negligible in dense sensor deployments, this approach can successfully timestamp the sample.

We illustrate this through an example. In Figure 10 let t_n^{iA} be the residence time at the i^{th} hop node and let t_p^{iA} be the propagation delay for the i^{th} hop. Then, the residence time of the sample from A is given by:

$$T_A = \sum_{i=0}^{i=3} t_n^{iA} + \sum_{i=1}^{i=4} t_p^{iA}. \quad (2)$$

Noting that propagation delay (of radio waves) incurred over several hundred meters (path distance to sink) is on the order of nanoseconds, we neglect the second summation in Equation 2. The time spent at a node is generally on the order of milliseconds and cannot be neglected. Under this assumption, T_A can be calculated by summing up the times spent at each node. As this packet reaches the base station S, the base station notes the time (its own local time) at which it received this packet say τ_A. Hence, the sample must have been generated at $\tau_A - T_A$ (T_A is obtained from the packet header) in the local time of the sink. The same procedure is applied for sample s_B. Now s_A and s_B can be aligned since $\tau_A - T_A = \tau_B - T_B$.

Our scheme eliminates many of the errors that time synchronization schemes have to contend with since we compute residence times close to the device [3, 4]. However, perhaps to a greater extent than those schemes, our scheme is impacted by clock drift. There are two problems brought about by clock drift. First, if the residence times are long (as they can be with our compression schemes, Section 5), then the timestamp can be significantly skewed. Second, clock drift can change the sample clocking, *i.e.*, individual samples may not be exactly 10ms apart when sampling at 100 Hz. The latter "problem" might be considered unimportant, since the device would be sampling the phenomenon correctly (when it happens), just not at the frequency it was supposed to. We return later to discuss the former problem.

In summary, our data time-stamping scheme incurs little overhead (a residence time field in every packet) and can be implemented easily as we now discuss.

6.3 Implementation

We have implemented this scheme on the Mica-2 platform. This section discusses the details of our implementation.

Since the default timer in TinyOS [6] doesn't provide an interface to read a 64-bit clock, we modified the Timer component in TinyOS to export a function (`getSysTime()`) to do this. Our modified timer component uses the "real-time clock" which ticks at a nominal rate of 32 Khz. Our 64-bit clock ticks at a 4Khz sampling rate (basically downsampling the real-time clock). This resolution is sufficient for our application, where the inter-sample time is on the order of 10ms.

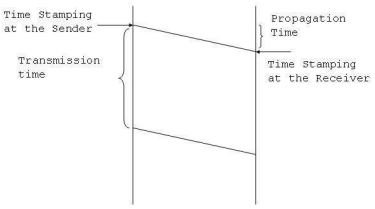

Figure 11: Time Stamping

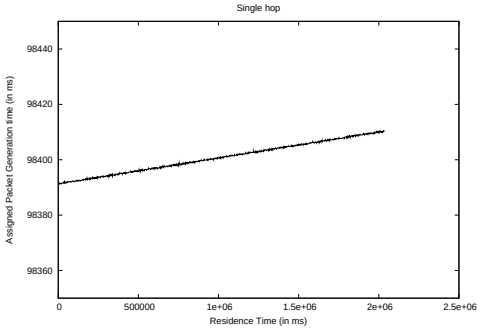

Figure 12: Time Drift - Single hop

We also modified the radio related components in TinyOS to timestamp the packet close to the source. We timestamp an incoming packet just after the reception of start symbol (for received packets) and just after the transmission of the timing bits (for transmitted packets) as shown in Figure 11. This brings the time-stamping as close as possible to the first byte transmission and the first byte reception of the packet. As others have observed [4], this eliminates the send, access, transmission, and reception time delay for the packet.

When a packet is being transmitted, just after it is timestamped, the radio layer makes an upcall to an application provided handler. In our case, this upcall allows Wisden to update the residence time (by adding the current value of the residence time, the difference between the current timestamp, and the time when the packet was received) of the packet, but our implementation is generic enough to allow other applications to do quick application-specific processing.

Finally, we also timestamp samples received from the vibration card. The vibration card periodically sends data samples in a message to mica2 motes over the UART. For samples received from the vibration card the time-stamping is done on mica2 at the reception of first byte of the packet[9].

[9]There are a couple of subtle issues with this. First, there may be some on-board latency on the vibration card that our scheme misses. Second, the vibration card uses its own clock for sampling, not the Mica2's clock, and the relative drift between them might induce error. We need to investigate these issues more carefully, but

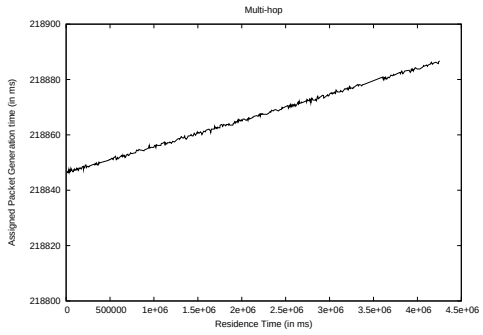

Figure 13: Time Drift - Multi hop

Rather than associate each sample in a packet (we use 80 byte packets which allow up to 18 samples) with a residence time, we only maintain the residence time of the *first sample*, and add an inter-sample time field so the base station can compute the time-stamp of the other samples.

6.4 Performance Evaluation

As we have discussed above, the primary determinant of accuracy in our system is clock drift. Our compression schemes suppress silence periods. During times of no activity, samples are not transmitted to the base station and remain within the network. Thus, their residence times are large, and the timestamps assigned to them can be affected by drift.

How much is the drift of the real-time clock on the Mica-2s? We used an oscilloscope to extract the clock signal from 7 different Mica-2s, then analyzed the dominant frequency in the signal. Table 2 shows the measured frequency of these Mica-2s. We see that the worst-case drift from the nominal clock rate is approximately 36 ppm and is consistent with the specifications in the clock chip's data-sheet [1]. However the relative drift between the motes is on the order of 10 ppm which matches with the value obtained in the next experiment.

Mote	Crystal Frequency
A	32766.8
B	32766.8
C	32767.1
D	32766.8
E	32766.7
F	32767.1
G	32766.7

Table 2: Measured frequency of MICA2 motes clock

To analyze the effect of clock drift of a single mote we performed the following experiment. We built a single hop network consisting of the base station and one mote. The mote time-stamps a sample at start up. It then sends *exactly* the same sample repeatedly. This computes the time-stamp assigned to a packet were it to be assigned different residence times in a one-hop network. Figure 12 plots the assigned packet generation time at the base station as a function of residence time. In the absence of drift, we would have expected a horizontal line. The slope of the line corresponds to a drift of 10ppm, which matches the relative drift we measured using an oscilloscope. We performed this experiment on five different sets of motes and obtained similar results.

Finally, we evaluate the impact of multi-hopping on our time-stamping technique. For this we repeated the previous experiment, but with 3 intermediate nodes between the sender and the base station. Each intermediate node delays the packet by 4 seconds before forwarding it (this mimics the behavior of Wisden, where intermediate nodes store a packet in an in-memory cache that is of fixed size). Figure 13 plots the assigned packet generation time at the base station as a function of residence time. Here, too, we find that the assigned sample generation time drifts with residence time by approximately 10 ppm.

If we assume 10 ppm to be a reasonable drift for the Mica-2s, this means that a sample can stay in the network for at most 1000 seconds (about 15 mins) before its timestamp accumulates more than 10ms error. This means that our run-length encoding must not accumulate silence periods longer than a few minutes. But there is another, more subtle, problem. If there is a long-lasting structural vibration event (say for several minutes), rate-limiting will cause these samples to accumulate significant residence times. To avoid this, we propose to piggyback on outgoing packets, one or two stored samples, so that their generation time can be accurately timestamped at the base station in *advance*. Wisden does not yet implement this. Finally, we'll note that our wavelet compression scheme is less susceptible to this drift because the low-resolution coefficients are sent immediately and incur low residence times.

7. DEPLOYMENT EXPERIENCE

Finally, as a proof-of-concept, we deployed a 10 node Wisden system on a test structure. This structure (Figure 14) resembles the frame of a hospital ceiling, about 40 ft. long and 20 ft. wide. When completed, this structure will have a shaker to impart forced vibrations, but it does not have one currently [10]. We instrumented this structure by affixing the accelerometers with heavy-duty double-sided tape (on the advice of a local structural engineer), and wrapped the rest of the assembly with gaffer tape. We then repeatedly hit the structure with a 2-by-4[11] for 20 seconds, also on the advice of our local expert.

Figure 15 is a screenshot of the collected sample data, aligned at the base station. The 10 motes formed a multi-hop network and transmitted all of the recorded vibration data back to base station within 5 minutes. The average residence time incurred by a packet in our experiment was 142 seconds; some of the delay can be attributed to the sustained excitation, and some to packet loss. Finally, testifying to the performance of the time synchronization scheme, we found that the *onset* time of one of our forced vibrations was within one sample time (actually 8ms) across all our accelerometers.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have described the design of a wireless structural data acquisition system called Wisden. The system mimics wired data acquisition systems, and incorporates novel reliable transport, time synchronization, and compression algorithms. In the next few months, we hope to gain significant experience with the system's overall accuracy and performance, by deploying it on several large

believe that because our timestamping requirements are relatively coarse (order of ms), they will not affect Wisden significantly.

[10] For testing the efficacy of our reliability scheme, we used our department office building in order to obtain realistic radio propagation. For the deployment test, where we needed to impart vibrations, it was more convenient to use this structure.

[11] Structural engineers have been known to use equally elegant methods (*e.g.*, driving a truck off a platform) to induce vibrations!

Figure 14: The Ceiling Structure

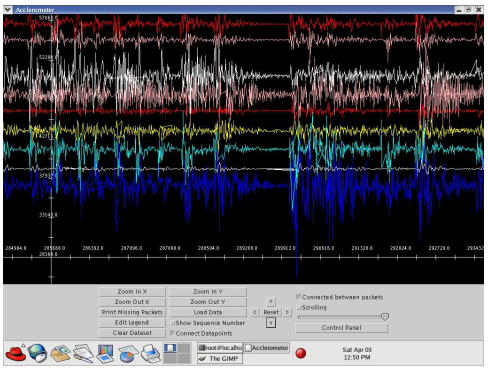

Figure 15: Data collected from the ceiling structure

structures at different scales. We intend to use this experience to evolve the system.

9. ACKNOWLEDGEMENT

We thank our shepherd Gul Agha and the anonymous referees for their constructive suggestions that improved the paper's presentation. We're grateful to Profs. John Caffrey and Sami Masri for providing access to the ceiling structure, Omprakash Gnawali for help during various stages of implementation, and to the members of the USC Embedded Networks Laboratory for feedback on the design of Wisden.

10. REFERENCES

[1] *Thin SMD Low/Medium Frequency crystal unit.*

[2] ELSON, J., AND ESTRIN, D. Time synchronization for wireless sensor networks. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium* (2001), IEEE Computer Society, p. 186.

[3] ELSON, J., GIROD, L., AND ESTRIN, D. Fine-grained network time synchronization using reference broadcasts. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)* (Boston, MA, December 2002).

[4] GANERIWAL, S., KUMAR, R., AND SRIVASTAVA, M. B. Timing-sync protocol for sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems* (2003), ACM Press, pp. 138–149.

[5] GANESAN, D., GREENSTEIN, B., PERELYUBSKIY, D., ESTRIN, D., AND HEIDEMANN, J. An evaluation of multi-resolution search and storage in resource-constrained sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*. (2003).

[6] HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D., AND PISTER, K. System archtecture directions for networked sensors. *SIGPLAN Not. 35*, 11 (2000), 93–104.

[7] INTANAGONWIWAT, C., GOVINDAN, R., AND ESTRIN, D. Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking* (Boston, MA, August 2000), ACM Press, pp. 56–67.

[8] KANG, T. H., RHA, C., AND WALLACE, J. W. Seismic performance assessment of flat plate floor systems. CUREE-Kajima Joint Research Program.

[9] LYNCH, J. P., SUNDARARAJAN, A., LAW, K. H., KIREMIDJIAN, A. S., AND CARRYER, E. Power-efficient data management for a wireless structural monitoring system. In *Proceedings of the 4th International Workshop on Structural Health Monitoring* (Stanford, CA, September 15-17 2003), vol. 1.

[10] MAINWARING, A., POLASTRE, J., SZEWCZYK, R., CULLER, D., AND ANDERSON, J. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications* (Atlanta, GA, 2002), pp. 88–97.

[11] MILLS, D. L. Internet time synchronization: The network time protocol, 1989.

[12] SANKARASUBRAMANIAM, Y., AKAN, O. B., AND AKYILDIZ, I. F. Esrt: event-to-sink reliable transport in wireless sensor networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing* (Annapolis, Maryland, USA, June 2003), pp. 177–188.

[13] STANN, F., AND HEIDEMANN, J. Rmst: Reliable data transport in sensor networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications* (Anchorage, Alaska, USA, April 2003), IEEE, pp. 102–112.

[14] TAUBMAN, D. S., AND MARCELLIN, M. W. *JPEG 2000: Image Compression Fundamentals, Standards, and Practices*. Kluwer Academic Publishers, 2001.

[15] VAN GREUNEN, J., AND RABAEY, J. Lightweight time synchronization for sensor networks. In *Proceedings WSNA, San Diego, California, USA.* (2003).

[16] VETTERLI, M., AND KOVACEVIC, J. *Wavelets and Subband coding*. Prentice Hall, New Jersey, 1995.

[17] WAN, C.-Y., CAMPBELL, A. T., AND KRISHNAMURTHY, L. Psfq: A reliable transport protocol for wireless sensor networks. In *Proceeding of First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002)* (Atlanta, September 2002), pp. 1–11.

[18] WAN, C.-Y., EISENMAN, S. B., AND CAMPBELL, A. T. Coda: Congestion detection and avoidance in sensor networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).* (Los Angeles, November 2003), pp. 266–279.

[19] WOO, A., AND CULLER, D. A Transmission Control Scheme for Media Access In Sensor Networks. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2001)* (2001).

[20] WOO, A., TONG, T., AND CULLER, D. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).* (Los Angeles, CA, November 2003).

[21] ZHAO, J., AND GOVINDAN, R. Understanding Packet Delivery Performance In Dense Wireless Sensor Networks. In *Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003).* (Los Angeles, CA, November 2003).