Wireless Sensor Networks for Structural Health Monitoring

by Sukun Kim

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor David E. Culler
Research Advisor

(Date)

* * * * * * *

Professor James W. Demmel
Second Reader

(Date)

Wireless Sensor Networks for Structural Health Monitoring

Abstract

Wireless Sensor Networks for Structural Health Monitoring

by

Sukun Kim
Master of Science in Computer Science
University of California at Berkeley
Professor David E. Culler, Research Advisor

A structural health monitoring system is designed, implemented, and tested using Wireless Sensor Networks (WSN). The project is targeting a deployment on the Golden Gate Bridge. Ambient vibration of the structure is monitored and is used to determine the health status of the structure. With WSN, low cost monitoring is possible without interfering with the operation of the structure.

To capture minute signals, a new accelerometer board is designed and the system is carefully calibrated against tilt and temperature. Signal processing is used to increase the quality of the sample. Sampling jitter is analyzed, and it is minimized. To collect data reliably, a bidirectional antenna provides longer connectivity, and a reliable data collection component achieves reliability with little overhead. Deployment at a footbridge showed the system is operating successfully, and the collected data matched theoretical expectation.

This work shows that WSN is capable of new applications at the opposite end of the application spectrum from conventional applications: high-fidelity, high-volume data sampling and collection.

Contents

List of Figures **iv**

List of Tables **vi**

1 Introduction **1**

1.1 Structural Health Monitoring . . . 2
1.2 Benefits from using Wireless Sensor Networks . . . 2
1.3 Challenges to Wireless Sensor Networks . . . 3
1.4 Overview of Solution . . . 4

2 Hardware **6**

2.1 Mote . . . 6
2.2 Accelerometer Board . . . 8
2.2.1 Accelerometers . . . 8
2.2.2 Noise Floor Test and Dynamic Test . . . 9
2.2.3 Tilting Calibration . . . 13
2.2.4 Temperature Calibration . . . 16
2.3 Antenna . . . 19
2.4 Power . . . 22

3 Software Architecture **24**

3.1 Overall Structure . . . 24
3.2 Data Sampling . . . 24
3.2.1 High Frequency Sampling . . . 25
3.2.2 Jitter Analysis . . . 27
3.2.3 Removing Jitter . . . 29
3.3 Data Collection . . . 33
3.3.1 Protocol . . . 34
3.3.2 Implementation . . . 36
3.3.3 Evaluation . . . 38
3.3.4 Findings . . . 44
3.4 Structure Monitoring Toolkit . . . 46

4 Data Analysis **47**
4.1 Signal Processing . . . 47
4.2 System Identification . . . 47

5 Results from Footbridge Deployment **49**
5.1 First Deployment . . . 49
5.2 Second Deployment . . . 53

6 Related Work **57**

7 Conclusion **59**

Bibliography **62**

A Schematics of Accelerometer Board **65**

B Structure Monitoring Toolkit (SMT) Commands **67**

List of Figures

1.1 High-level Overview of the System 5

2.1 Hardware Block Diagram 7
2.2 Mica2 7
2.3 Accelerometer Board 8
2.4 Noise Level at Various Places 10
2.5 Time Series Data of Vault Test 11
2.6 Shake Table Test (0.5Hz), ADXL 202E 12
2.7 Shake Table Test (0.5Hz), SD 1221L 12
2.8 Shake Table Test (Increasing Frequency with Same Displacement), ADXL 202E 13
2.9 Shake Table Test (Increasing Frequency with Same Displacement), SD 1221L 14
2.10 YUASA used for Tilting Test 15
2.11 Sample Result of Tilting Test 15
2.12 Time Series of Temperature 16
2.13 Time Series of Acceleration. Thick line is SD 1221L. The other line is ADXL 202E. 17
2.14 Temperature and Acceleration 17
2.15 Estimated Parameters for Linear Regression 18
2.16 Deployment Plan for the Golden Gate Bridge 19
2.17 Bidirectional Patch Antennas 20
2.18 Antenna Installation on the Golden Gate Bridge 21
2.19 (Left) Current Board Design, (Right) Board Design Proposal 23

3.1 Overall Software Architecture 25
3.2 Source of Jitter 26
3.3 Occurrence of Jitter 27
3.4 (Left) One Atomic Section, (Middle) Multiple Atomic Sections, (Right) Multiple Atomic Sections with CPU Sleep 28
3.5 Time Series of Jitter at 1KHz 29
3.6 Time Series of Jitter at 5KHz 30
3.7 Time Series of Jitter at 6.67KHz 30
3.8 Histogram of Jitter at 1KHz 31

3.9 Histogram of Jitter at 5KHz 32
3.10 Histogram of Jitter at 6.67KHz 32
3.11 State Diagram of Sender 34
3.12 State Diagram of Receiver 35
3.13 Range of Radio Interference 36
3.14 Test result on Mica2 testbed 39
3.15 Example of Graph (Success Rate versus Packet Interval) used to measure maximum bandwidth 40
3.16 Payload and Packet Throughput 41
3.17 Packet Throughput in Pie Graph 42
3.18 Effective Bandwidth. 33% is usable 42
3.19 Graph showing where time is spent. Total time is 212.7μs 43
3.20 Usage of Packet Buffer 45

4.1 Imperfect Low-pass Filter 48

5.1 Footbridge Over I-80 50
5.2 Location of Nodes at the First Deployment 50
5.3 First Vertical Mode 51
5.4 Second Vertical Mode 51
5.5 First Horizontal Mode 52
5.6 Second Horizontal Mode 52
5.7 Location of Nodes at the Second Deployment 54
5.8 First Vertical Mode at the Second Deployment 54
5.9 Second Vertical Mode at the Second Deployment 55
5.10 Beams Supporting Span 56

List of Tables

2.1 Two Accelerometers 9
2.2 Power Consumption in Various Operational Situations (9V) 22

3.1 Maximum Channel Capacity and Measured Usable Capacity 40

5.1 Maximum Channel Capacity and Measured Usable Capacity 53

Acknowledgements

Special thanks to Professor David Culler who gladly discussed numerous challenges confronted in the project. Many thanks are owed to Shamim Pakzad for hardware, deployment and data analysis. Thank you to Professor James Demmel for calibration process and data analysis. And thanks to Professor Gregory Fenves and Professor Steve Glaser for insights about structure and help for many perspectives of the project. Thanks to Tom Oberheim for the accelerometer board, and thanks to Alan Broad for debugging the board. Martin Turon ran the oven test for temperature calibration and arranged the tilting test. And thanks to Rob Szewczyk for radio, antenna, enclosure, jitter analysis and numerous other suggestions. Crossbow Technology generously donated the assembly of the accelerometer board, and provided equipment for tests.

This work is supported by the National Science Foundation under Grant No. EIA-0122599.

Chapter 1

Introduction

Wireless sensor networks (WSN) enable many monitoring applications. Habitat monitoring at Great Duck Island [26] is an example. Wireless sensor networks monitor the microclimates in and around nesting burrows used by the Leach's Storm Petrel. Their goal is to develop a habitat monitoring kit that enables researchers worldwide to engage in the non-intrusive and non-disruptive monitoring of sensitive wildlife and habitats. Deployment in the Redwood forest is also a good example. A WSN is deployed on redwood trees, and environmental data around the tree is gathered. These typical usages of WSN are for cases with low data rate, small data size, low duty cycle, and extreme emphasis on low power consumption.

There is a different class of applications at the opposite end of application spectrum. Structural health monitoring of buildings, bridges and other structures requires high data rate, large data size, and a relatively high duty cycle. In this class, the focus is more on high fidelity of data, and aggressive sampling and collection rather than low power consumption. This work is targeting the Golden Gate Bridge, but most of parts are applicable to structural health monitoring in general. First, structural health monitoring will be introduced, in general. Benefits in using WSN will be discussed. Then, we will see what is required for structural health monitoring, and challenges to WSN. At last, a high level overview of our solution will be provided.

1.1 Structural Health Monitoring

We will look at structural health monitoring (SHM) in more detail. Structural health monitoring is estimating the state of structural health, or detecting the changes in structure that effect its performance. Two major factors are time-scale of change and severity of change. Time-scale is how quickly the change occurs, and severity is the degree of change. Two major categories of SHM are disaster response (earthquake, explosion, etc.) and continuous health monitoring (ambient vibrations, wind, etc.). This work is focused on continuous health monitoring. There are two SHM approaches: direct damage detection(visual inspection, x-ray, etc.) and indirect damage detection (change in structural properties/behavior). We use indirect detection, especially through vibration.

1.2 Benefits from using Wireless Sensor Networks

Structural health monitoring itself is not a new concept. The conventional method uses PCs wired to piezoelectric accelerometers [23]. However, this method has drawbacks in that (1) wires have to run all over the structure, so this method may disturb the normal operation of the structure, (2) the cost of equipment is high, (3) installation is very expensive due to wiring, and (4) its maintenance is also expensive. Compared to the conventional method, WSN provides the same functionality at a much lower price which permits much denser monitoring.

First let us look at the cost of hardware. A conventional system with a PC and piezoelectric accelerometers is used as a reference. It costs \$40,000 per sampling point. Our system costs \$600 per point. In WSN, wiring is not needed, so installation and maintenance are easy and inexpensive. Moreover, WSN affect the operation of the structure less.

The advantage of WSN based structural health monitoring can be amplified by MEMS accelerometers. Since the MEMS accelerometer is a silicon chip, it is very compact in size, consumes little energy, and is inexpensive. Without MEMS accelerometer, WSN's small size, low power and low cost will be degraded.

1.3 Challenges to Wireless Sensor Networks

For structural health monitoring, we need to read acceleration signals down to 500μG (1G is the gravity) [12], at a frequency higher than 1KHz synchronously at all nodes. And we need all data.
In addition to these real-time, high fidelity performance requirements, there are other ones. Monitoring needs be economical. The cost includes the system itself, installation, and maintenance. We do not want to disturb the structure being monitored, and introduce no hazards.

The requirements of structural health monitoring can be challenging to WSN.

- **High accuracy of sample**: It means the final reading needs to detect signals down to 500μG without significant distortion. Sources of distortion include the noise floor of the system (including accelerometer, amplifier, analog to digital converter, etc), installation error, and temperature variation.

- **High-frequency sampling**: This implies low jitter. Jitter is a variation in sampling intervals.

- **Time synchronization**: Sampling needs to start at the same time on all nodes although the sampling should be done over multiple nodes across the entire network. Furthermore, this needs be doen in spite of differences in drift of each clock. Otherwise, shifts in signals between different nodes can give a distorted picture of the structure.

- **Large-scale Multi-hop network**: In case the structure spans a long distance (e.g. 1 mile) like a bridge, it is impossible to cover the entire structure with single hop communication. So a large-scale multi-hop network is necessary to provide connectivity.

- **Reliable command dissemination**: If we fail to start some nodes, we will miss data for those points. Then we will have an imperfect picture of the bridge, which makes analysis very difficult or impossible.
- **Reliable data collection**: Not only commands, but also data needs to be transferred reliably. Missing samples make the analysis hard or even impossible.

Conventionally, the usage of WSN has been focused on low duty cycle, low data rate, moderately loss-tolerant sampling. Structural health monitoring is at the other end of application spectrum, and has different requirements. With less concern about energy, it needs high duty cycle, high frequency high-fidelity sampling, and reliable collection of large amount of data. These challenges are new to WSN.

1.4 Overview of Solution

Figure 1.1 shows high-level overview of the system. Operation can be decomposed into three phases.

- **Data sampling**: sample the vibration data of the structure, and logs it into EEPROM.
- **Data collection**: transfers data reliably to an external computing resource.
- **Data Analysis**: runs analysis algorithm, and determines health status. Sends feedback to nodes if needed.

The system can also be decomposed into its structural pieces. This paper is structured according to the components of the system. The system can be decomposed into three pieces: hardware, software on the node, and analysis software on a central computer. The following three chapters will cover these three components. Hardware issues including mote,

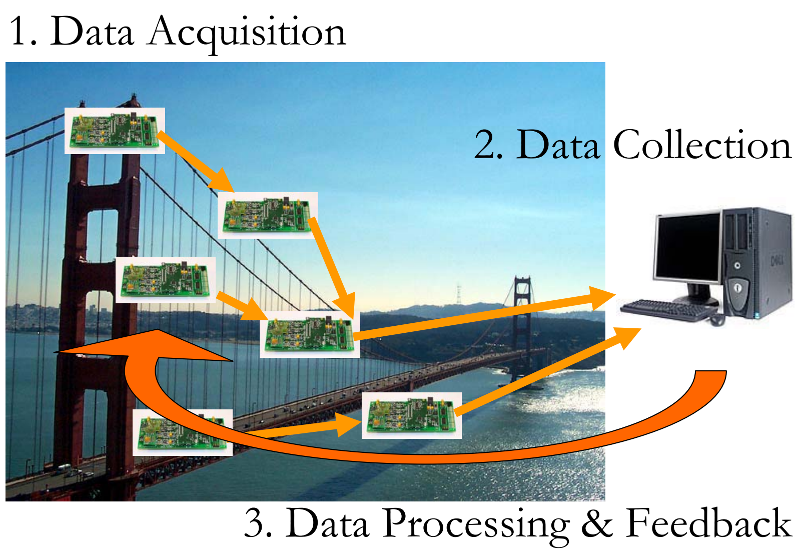


Figure 1.1: High-level Overview of the System

accelerometer board, antenna and power will be presented in Chapter 2. Software structure, data sampling software and data collection software are covered in Chapter 3.

Then data analysis method is described in Chapter 4, and results from real world deployments will be shown in Chapter 5. Related work, and conclusions will follow in Chapters 6 and 7.

Chapter 2

Hardware

This chapter explains hardware issues in more detail. Figure 2.1 shows an overview of the hardware as a blcok diagram. Accelerometer board has sensors and signal processors (low-pass filter, analog to digital converter). A mote stores data from the accelerometer boards, and later sends the data through an antenna. Mote, accelerometer, antenna and power issues will be discussed. For each component, the problem statement, solution and evaluation will be presented within each section.

2.1 Mote

The mote drives the accelerometer board, stores data, and communicates data.

The Mica2 [11] is used for this work. The Mica2 is shown in Figure 2.2. The main components of ths mote are the microcontroller, radio chip, and external data storage. The microcontroller is ATmel ATmega 128L. It has 128KB of program memory, 4KB of RAM, and runs at 8MHz. The radio chip is the Chipcon CC1000. Its data rate is 38.4Kbaud. We used a version of Mica2 with a radio frequency of 433MHz. The Mica2 also has an external 512KB EEPROM for data storage.

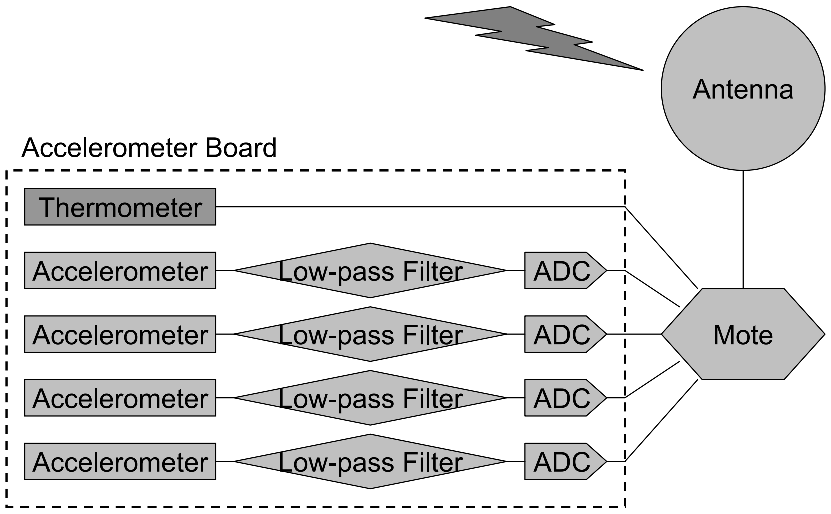


Figure 2.1: Hardware Block Diagram

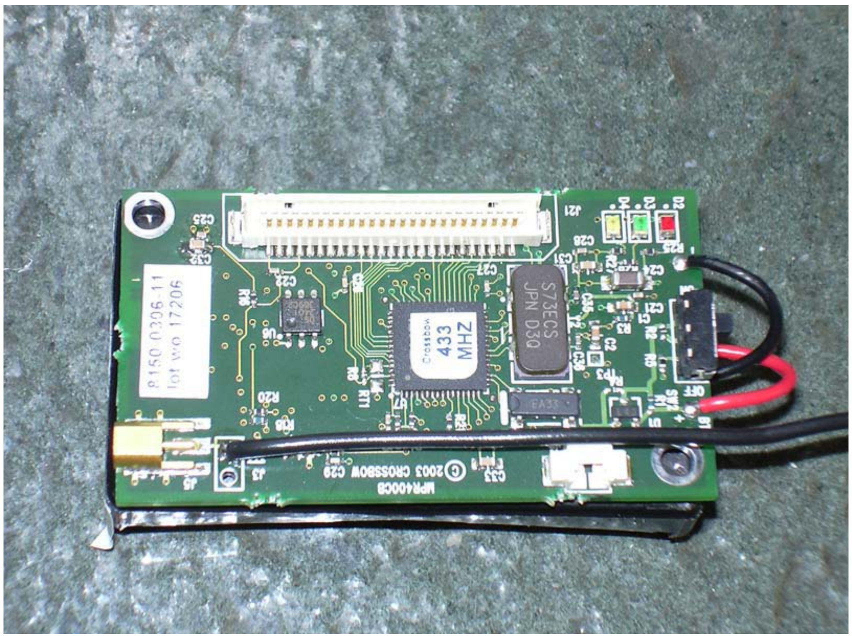


Figure 2.2: Mica2

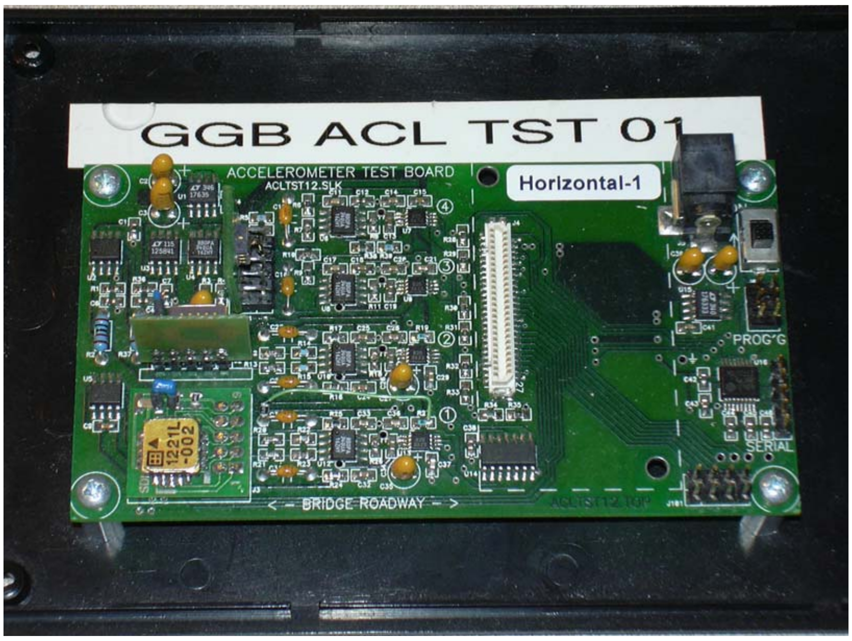


Figure 2.3: Accelerometer Board

2.2 Accelerometer Board

A new accelerometer board is designed for structural health monitoring. It has very accurate accelerometers and a thermometer to calibrate them. It is shown in Figure 2.3, and its schematics is in Appendix A.

2.2.1 Accelerometers

Two quite different sources of vibration (earthquake and ambient) are monitored using a single board. So the accelerometer board has two kinds of accelerometers: ADXL 202E and Silicon Designs 1221L. Table 2.1 shows characteristics of each accelerometer part (range and system noise floor shows the performance in combination with the entire system). The accelerometer board contains one of ADXL 202E, and two of Silicon Designs 1221L, and four of 16 bit analog to digital converters (ADC). On the span of the bridge, we planned to measure acceleration in two directions perpendicular to the bridge span: one up-down, one

Table 2.1: Two Accelerometers

	ADXL 202E	Silicon Designs 1221L
Type	MEMS	MEMS
Number of axis	2	1
Range of System	-2G to 2G	-0.1G to 0.1G
System noise floor	$200(G/\sqrt{Hz})$	$30(G/\sqrt{Hz})$
Price	\$10	\$150

across the span. The bridge span does not move much parallel to the span, and its signal is not interesting. And for the tower, we measure two directions parallel to the ground: along the span and across the span. The tower does not have much movement in a direction vertical to the ground. Acceleration is measured both by the ADXL 202E and Silicon Designs 1221L. Since the ADXL 202E has two accelerometers in directions perpendicular to each other, one ADXL 202E chip captures both directions. The Silicon Designs 1221L has one axis, so two of them are used. Thus, there are four accelerometer channels in total. The ADXL 202E channels have a range from -2G to 2G, to capture big movements like an earthquake. The Silicon Designs 1221L has a narrower range (from -0.1G to 0.1G) to sample subtle signals like ambient vibration. The vertical channel of the Silicon Designs 1221L has a 1G offset to compensate for gravity.

2.2.2 Noise Floor Test and Dynamic Test

To see static characteristic of accelerometers, the accelerometer board was put in a quiet vault, which is isolated from outside vibration and sound with constant temperature. The vault is located in Lawrence Berkeley National Laboratory, and is used to detect earthquakes and nuclear weapon development. Figure 2.4 shows how quiet the inside of the vault is (solid line) compared to normal office environment (small dashed line). And it also shows a reference reading from a very sophisticated accelerometer in the vault, which

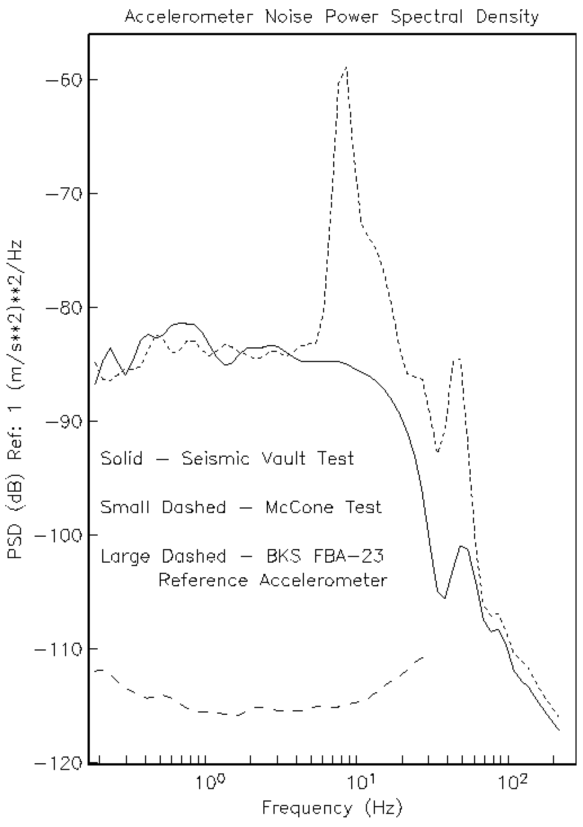


Figure 2.4: Noise Level at Various Places

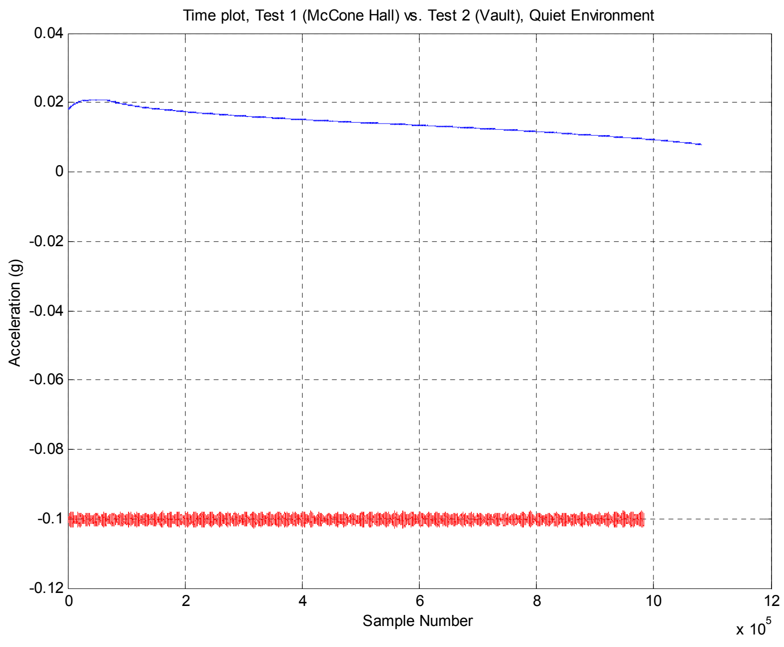


Figure 2.5: Time Series Data of Vault Test

is used for seismic research (large dashed line). The system with Silicon Designs 1221L has a noise level over 20dB higher. Figure 2.5 shows the time plot of acceleration in the vault and office environments for a 30 minute period. The data is not calibrated against the temperature. The lower line shows noise in a normal office environment. We can see noise from machines, which is also visible in Figure 2.4. The upper line shows acceleration readings from the vault. Drift is observed in this case. On the test day, it was rainy and cold outside, and it was warm at the inside of the vault. We put the accelerometer in the vault, and immediately started sampling. So the accelerometer board experienced a drastic change in temperature. The drift is almost 10mG, which is significant compared to the noise floor, and sensitivity. More discussions on temperature will follow later.

To see the dynamic behavior of accelerometers, we performed a shaking table test with constant temperature. Even though the test site was not completely free from vibration and sound noise, it was quiet enough for the dynamic range of the shaking table

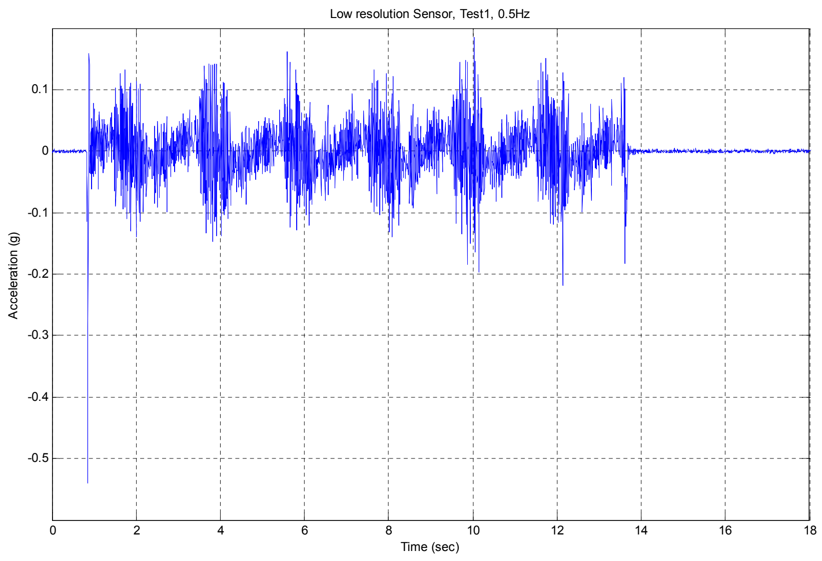


Figure 2.6: Shake Table Test (0.5Hz), ADXL 202E

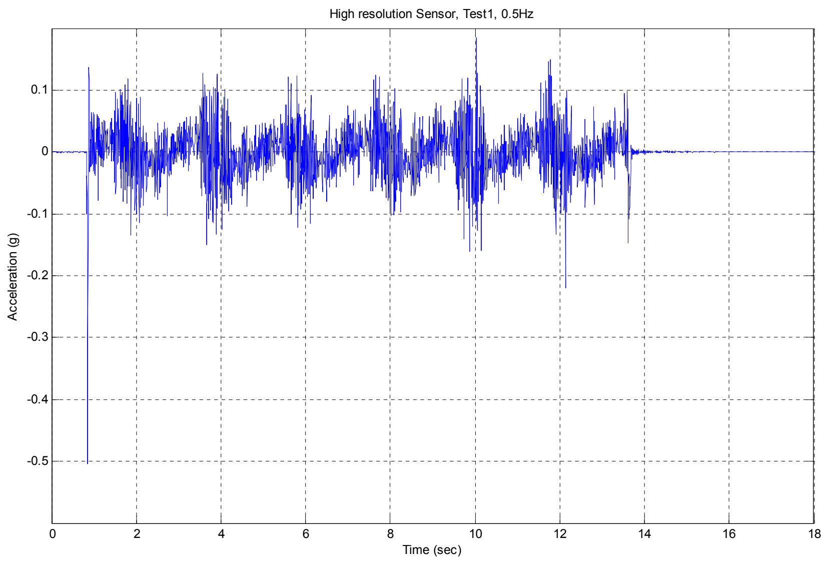


Figure 2.7: Shake Table Test (0.5Hz), SD 1221L

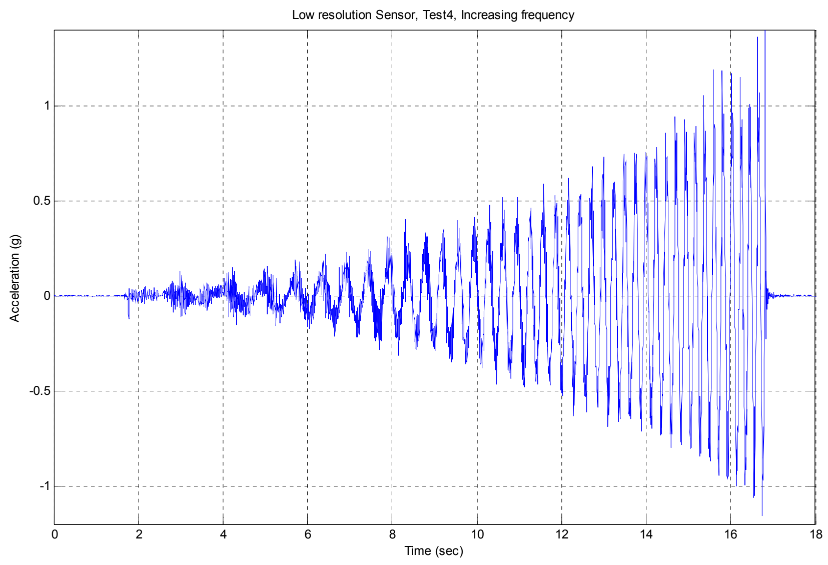


Figure 2.8: Shake Table Test (Increasing Frequency with Same Displacement), ADXL 202E

to dominate noise. Results are shown in Figures 2.6, 2.7. Figure 2.6 is the result of ADXL 202E, Figure 2.7 is the result of Silicon Designs 1221L, and the driving frequency is 0.5Hz. Data are read from both channels at the same time. We can see Silicon Designs 1221L has less vibration noise in a static situation (when there is no movement). Figures 2.8, 2.9 show another experiment on the shaking table. Here the frequency increases while displacement remains constant. When the amplitute of the movement gets large, signals from the Silicon Designs 1221L has its peaks clipped. It seems like Silicon Designs 1221L has larger damping factor than ADXL 202E, so that sharp surges at peaks are damped.

2.2.3 Tilting Calibration

There exist variations in accelerometers, analog filter components, ADCs, etc. Therefore, we need to calibrate the acceleration measurement system (not only the accelerometer) to correctly convert data to acceleration. A tilting test measures data at various angle points and matches the angle and the data actually read. Thanks to gravity,

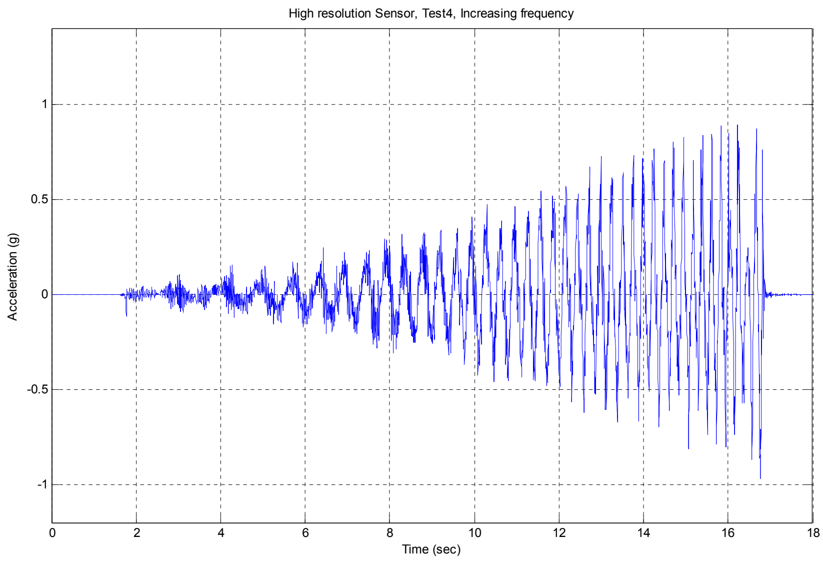


Figure 2.9: Shake Table Test (Increasing Frequency with Same Displacement), SD 1221L

we can create arbitrary accelerations from -1G to 1G. Through this test we can measure the offset and the gain of the system for each node.

We used the tilting calibration equipment YUASA as shown in Figure 2.10. YUASA automatically changes angle. The accuracy of the angle is within 0.0005 degree which is 8.73μG near 0 degree and 0.0381nG near 90 degree. The mote continuously sends data. LabView [8] is running on the PC, and reads an angle from the YUASA and reads measured data in data packets from the mote. Then it stores them as a table to a file. A sample of data is shown in Figure 2.11. The top figure is time series of accelerometer readings from all channels. Other figures below show real angles versus accelerometer readings. As can be seen in the figures, the reading shows normal distribution, and its average is very linear to acceleration.

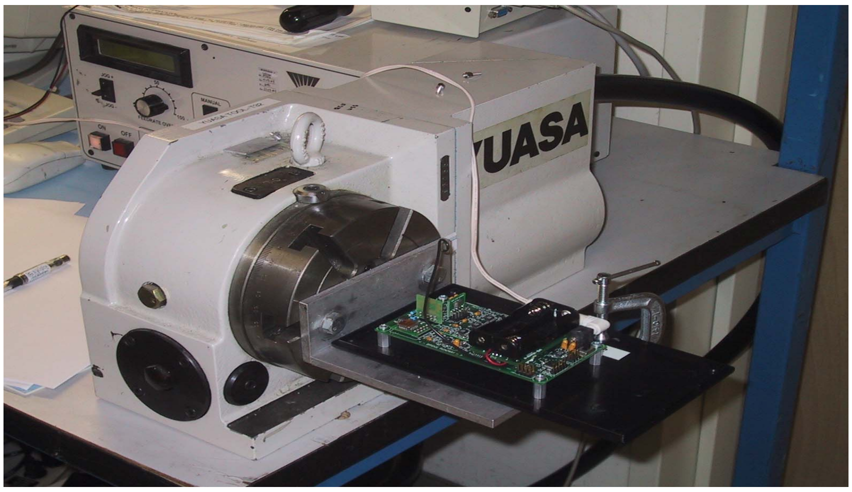


Figure 2.10: YUASA used for Tilting Test

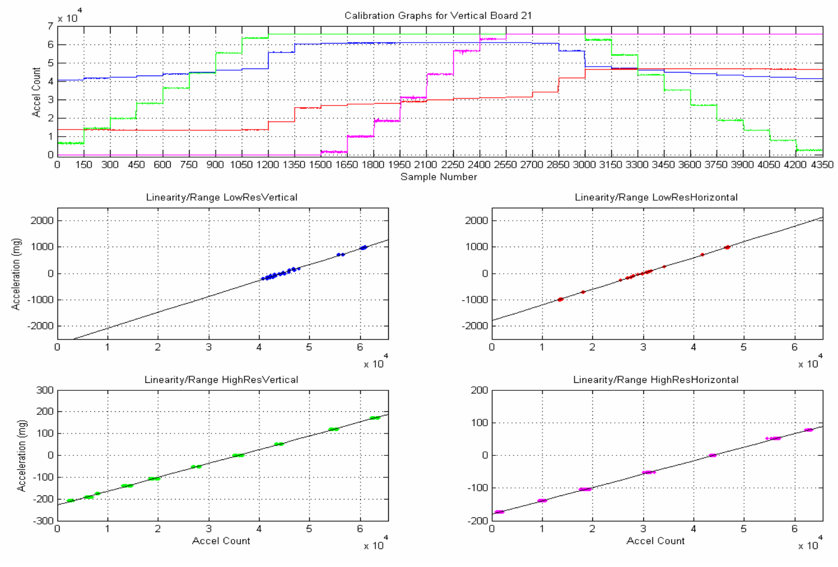


Figure 2.11: Sample Result of Tilting Test

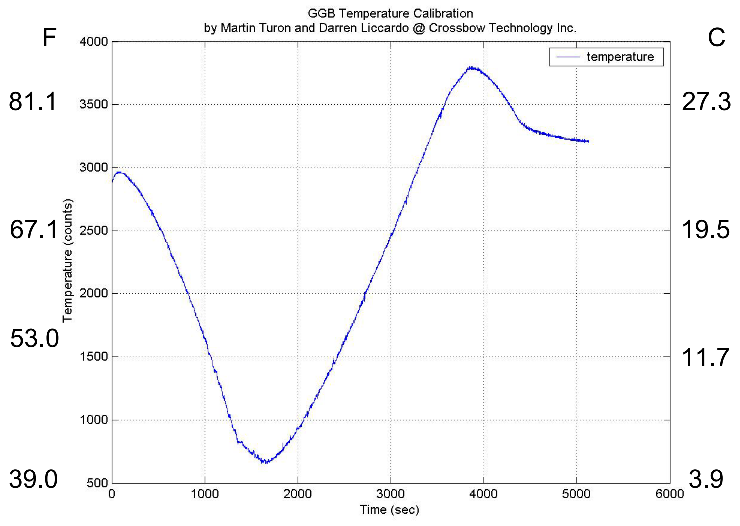


Figure 2.12: Time Series of Temperature

2.2.4 Temperature Calibration

Accelerometers are very sensitive to temperature change. The bridge is exposed to significant variation in the temperature. Temperature can change up to 45^{o}F within a day. From our calibration measurements this can shift the signal by 40mG in a sensitive channel. Recall we are looking at very subtle signal, sometimes down to 500μG. The signal to noise ratio in this particular case is 1.25%, which means signal is totally dominated by noise from temperature variation. Therefore, we need to calibrate the acceleration data against temperature. The accelerometer board contains a temperature sensor: Microchip TC77, which is used for calibration.

We performed temperature a calibration test to see how well accelerometers can be calibrated against temperature using the external thermometer on the board (Microchip TC77). We first put the accelerometer board into an oven, changed temperature, and associated temperature and accelerometer data. Figure 2.12 shows temperature change over time, and Figure 2.13 shows acceleration change. The SD 1221L horizontal channel

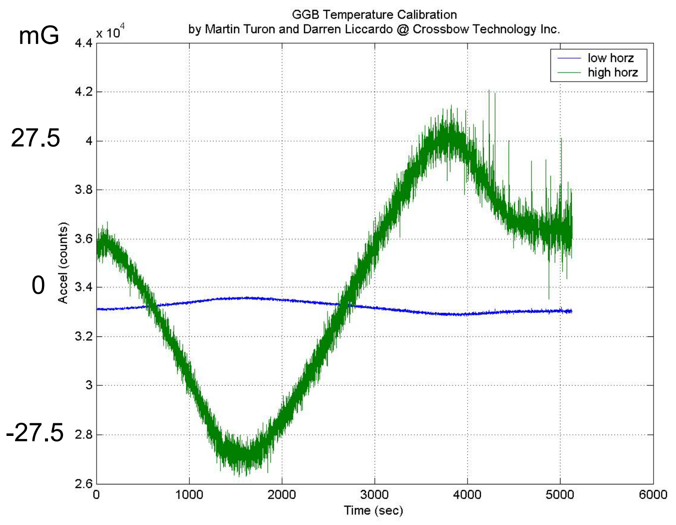


Figure 2.13: Time Series of Acceleration. Thick line is SD 1221L. The other line is ADXL 202E.

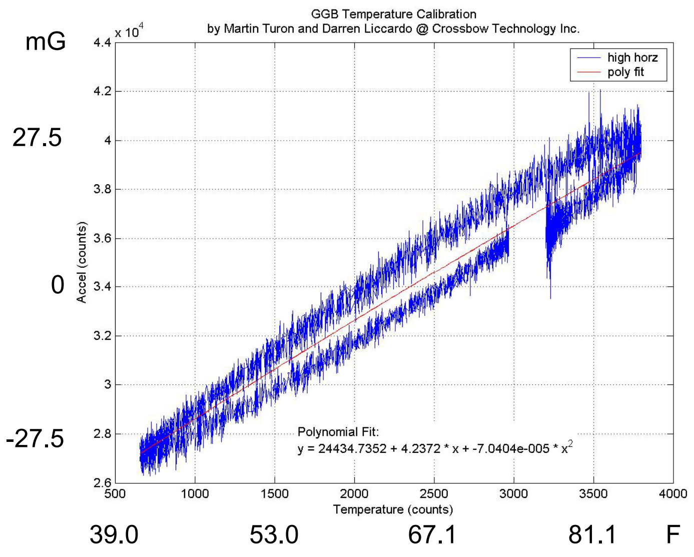


Figure 2.14: Temperature and Acceleration

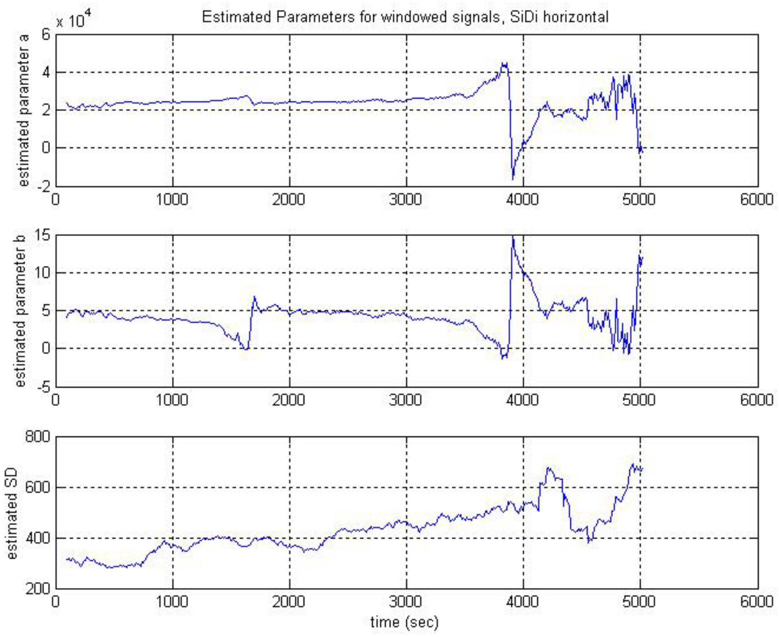


Figure 2.15: Estimated Parameters for Linear Regression

is shown, which suffers the most significant effect from temperature. The combined result is shown in Figure 2.14 (acceleration versus temperature). Results for the vertical channel and results for the ADXL 202E are very similar, except for the amplitude of change in acceleration.

We tried a linear regression model. A window of length 199 samples is considered and a linear regression model of the form

$$(Acceleration) = (RawAcceleration) + a + b * (Temperature) + e$$

is fit to that windowed segment of the data.

The Figure 2.15 shows plots of the estimated parameters for data shown in Figure 2.14 (SD 1221L horizontal channel). The parameters include a and b (as defined above) and the standard deviation of e, the error term in the regression model. There are strange peaks. They occur when the direction of temperature change reverses: from up to down, or from down to up.

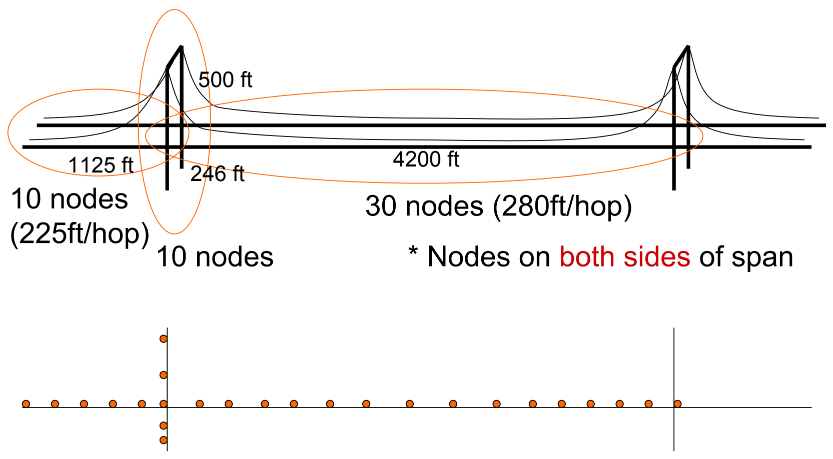


Figure 2.16: Deployment Plan for the Golden Gate Bridge

This test raises a new concern: hysteresis. The thermometer responds slower than accelerometer. We can't get rid of temperature effect simply by linear regression. However, considering the enormous rate of temperature change (1 degree Celsius per minute) in the test, estimated standard deviation of 3mG is quite good. We are investigating how significant this effect will be in a bridge environment. And we can also use the thermometer inside the accelerometer chip to avoid possible hysteresis.

2.3 Antenna

This project is ultimately targeting Golden Gate Bridge. Figure 2.16 shows the deployment plan. The longest hop is 280 ft. With a wire whip antenna, communication success rate is too low at this distance. Therefore, we looked at other antenna options to provide connectivity. Omnidirectional antennas are poorly suited to this application, instead we want a bidirectional antenna. This situation is not uncommon in structure

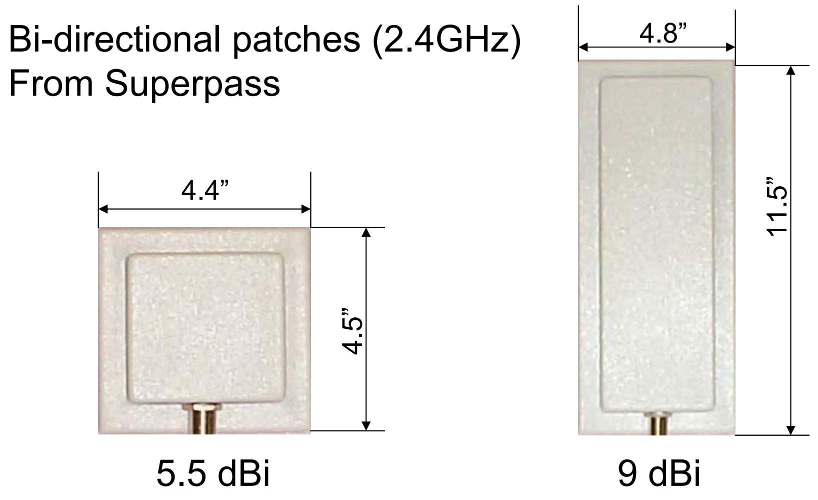


Figure 2.17: Bidirectional Patch Antennas

monitoring. Covering big bridges and buildings is not possible with bulky whip antenna. A patch antenna is reasonably small in size and increases range moderately orthogonal to the patch. However, there is a limitation for availability. Only a 2.4GHz product exists for this type of antenna, which are designed for WLAN operating at 2.4GHz. This means our target mote Mica2 is not compatible with the antenna. One possibility is to use the the MicaZ platform. The MicaZ is the same as the Mica2 except that it uses a standard IEEE 802.15.4 radio running at 2.4GHz.

We tested bidirectional patch antennas manufactured by Superpass [9], as shown in Figure 2.17. Figure 2.18 shows the installation on the Golden Gate Bridge. The installation is the same as the planned real installation on the bridge. A MicaZ was used for the test (just for the radio test, not for sampling the vibration data). A 9 dBi bidirectional patch is used for both side of communication. Radio power is set to maximum. At 150ft, packet success rate is 99.5%. At 200ft, success rate is 62.0%, which is prohibitively low. Adding a repeater will

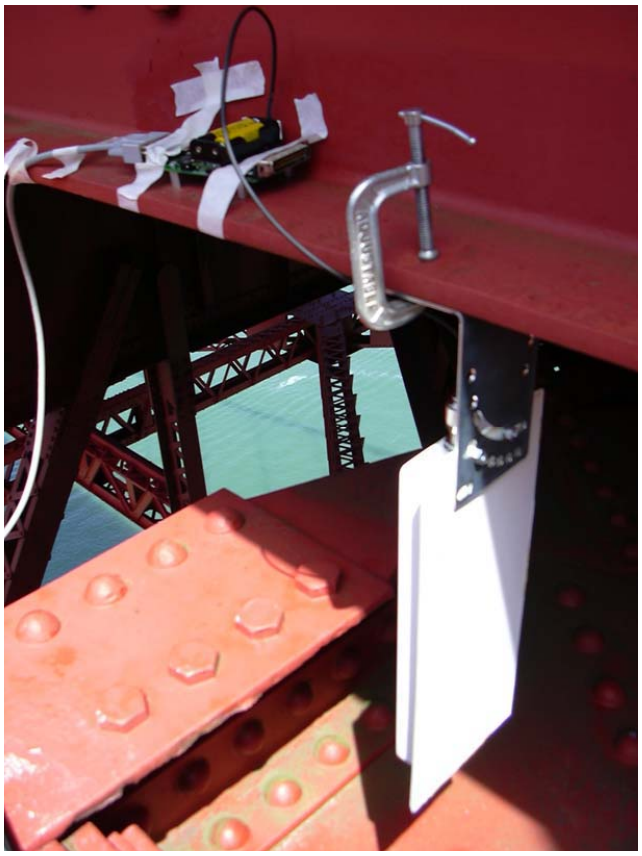


Figure 2.18: Antenna Installation on the Golden Gate Bridge

Situation	Consumption (mA)
Board Only	26.7
Idle	39.8
One LED On	42.6
Erasing Flash	74.7
Sampling	39.8
Transferring Data	43.2

Table 2.2: Power Consumption in Various Operational Situations (9V)

increase the success rate, but adding more hops increases collection time. A larger antenna could be used instead. But a large antenna may not fit into the small space for installation. More analysis on repeaters is future work.

2.4 Power

It is not always possible to have wired power at the deployment. Even if possible, the cost overwhelms its benefit. In a bridge environment, this is a problem. So the system needs to rely on a battery as a power source. We analyzed whether the design meets our desired length of deployment, and how much it would cost for the system to last as long as we desire.

Table 2.2 shows a power consumption profile of the system, which includes an accelerometer board and a mote. We can notice that the power consumption of the board itself is very high compared to that of the mote. The left side of Figure 2.19 shows the reason. Power is directly connected to the mote, sensor, and ADC. To run the mote, all other hardware components have to be powered on. The right side of Figure 2.19 shows a proposal for lower energy consumption. Only the mote is directly connected to the battery, and all other components can be powered off by mote. This will significantly cut power consumption by

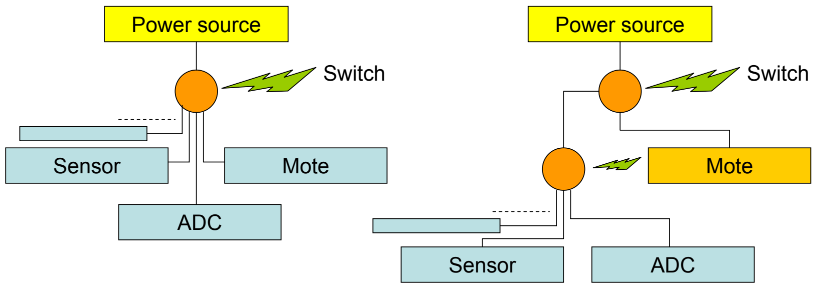


Figure 2.19: (Left) Current Board Design, (Right) Board Design Proposal

powering the accelerometer board only in the sampling phase.

To estimate how much energy is needed for a 3 week deployment, we assumed an operational model. The mote transfers data one third of a time, and stays in the idle state otherwise. Since sampling is short, and its power consumption is very close to that of idle state, sampling mode is not considered. We found a high-capacity lithium battery: the Tadiran 5930 [10] supplies 3.6V and contains 19Ah in a D size package. It costs $17 each. With 3 of these Tadiran 5930, the system will lasts 23 days, good enough for a 3 week deployment.

Chapter 3

Software Architecture

As an underlying software infrastructure, TinyOS [16] is used. TinyOS is an operating system developed in UC Berkeley. It is a de facto standard operating system in WSN. In this chapter, our findings and newly added components will be introduced.

3.1 Overall Structure

Figure 3.1 shows the overall structure of software. Structure Monitoring Toolkit (SMT) is an application layer program, which drives all components. On top of best-effort one-hop communication, broadcast is used for command dissemination, and MintRoute [27] is used for information reply. MintRout provides best-effort multi-hop convergence routing. Reliable data collection layer lies above broadcast and MintRout. SMT uses them all: broadcast, MintRout, reliable data collection. For time synchronization, FTSP [22] is used. BufferedLog [5] is used to support high frequency sampling.

3.2 Data Sampling

As presented in Chapter 1, there were 6 challenges. High accuracy of sample, high-frequency sampling, and time synchronization are related to data sampling. From

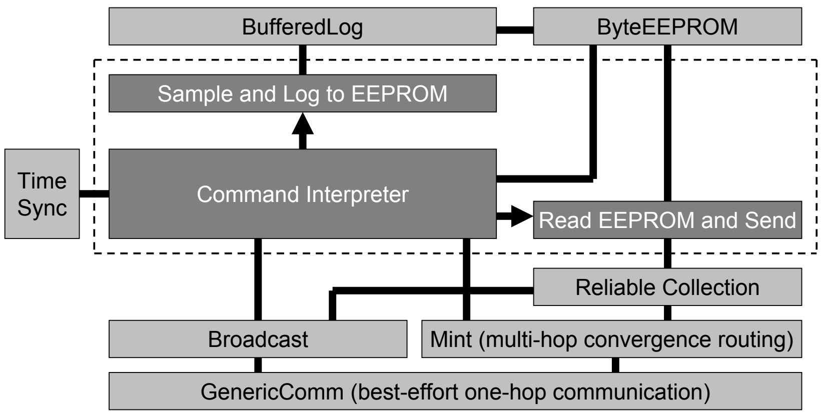


Figure 3.1: Overall Software Architecture

software's point of view, high accuracy of sample is a topic related to averaging as a signal processing step, which will be covered in Chapter 4, Data Analysis. Time synchronization is provided using FTSP. So high-frequency sampling will be mainly discussed here.

3.2.1 High Frequency Sampling

The fundamental frequencies of most structures are below 10Hz (A structure with higher fundamental frequencies would be too stiff). However by Nyquist theorem, sampling rate should be at least twice of that. Moreover, to reduce effect of noise, averaging is used (more details will be explained in Chapter 4), and the sampling rate should be multiplied by the number of samples averaged. All these factors increase the sampling rate close to KHz level. Structure monitoring requires regular sampling with uniform interval, and jitter becomes a critical problem as sampling rate gets higher.

There are two kinds of sources to jitter, and they are shown in Figure 3.2. Temporal jitter

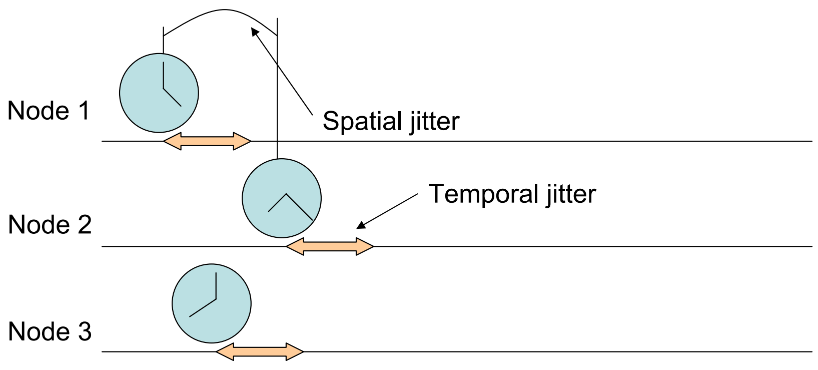


Figure 3.2: Source of Jitter

occurs inside a node, because actual sampling does not occur at a uniform interval. So even with only one node, temporal jitter happens. Spatial jitter between different nodes happens because of variation in hardware, and imperfect time synchronization. Even if two nodes agree to sample at time T, this T occurs at different absolute times for those two nodes. Spatial jitter occurs only when there are more than one node.

We assumed sampling frequency does not go beyond 200Hz, and 5% of total jitter (250μs) across system is assumed to be acceptable. Actually the tolerance of algorithm to jitter is not very well understoold. 5% is a conservative speculation. Time synchronization component FTSP provides 67μs error over 59-node 11-hop network [22], so spatial jitter is not a problem. The temporal jitter will be smaller than the spatial jitter, because causes of the spatial jitter include all the causes for the temporal jitter. However, when a sampling starts, more activities occur and more sources of jitter become active. Therefore, temporal jitter at this point can be worse than spatial jitter without sampling activity. Therefore,

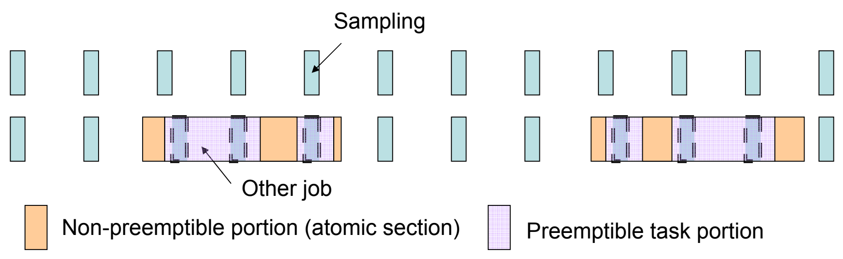


Figure 3.3: Occurrence of Jitter

here temporal jitter is considered.

3.2.2 Jitter Analysis

Figure 3.3 shows interaction of sampling and other jobs such as writing data from RAM to flash. Timer event for sampling occurs regularly with a uniform interval. However, to be serviced by the CPU, the CPU should finish servicing pending atomic section. Only then, can the CPU handle timer event and sampling. Polling is used in driving ADC. Therefore, the moment timer event occurs, sampling happens capturing the most recent sample.

Assume only up to one event can reside in the task queue when a timer event occurs (this assumption will be explained in the next subsection). Let N be the number of atomic sections. And C be the context switch time when timer event occurs while CPU is executing preemptible section. We assume that C is constant regardless of the portion of code running.

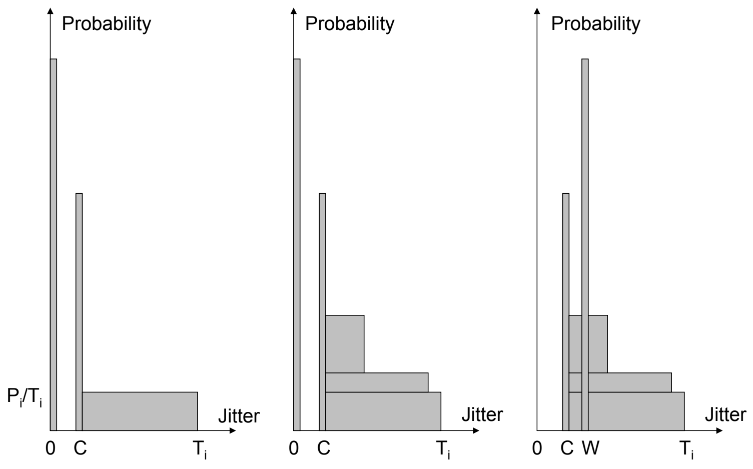


Figure 3.4: (Left) One Atomic Section, (Middle) Multiple Atomic Sections, (Right) Multiple Atomic Sections with CPU Sleep

And let T_i be the length of atomic section i. And P_i be the probability of atomic section i running on CPU when a timer event occurs. Let $X(i)$ be a random variable which is the remaining execution time of atomic section i running on CPU when a timer event happens. We assume $X(i)$ is uniformly distributed from 0 to T_i.

First, assume $N = 1$. Left figure of Figure 3.4 shows distribution of jitter. Peak at 0 is the case where there is nothing running on CPU when a timer event occurs. Peak at C is the case where preemptible code is running when a timer event occurs. And above C, there is a case where atomic section i is running on CPU when a timer event occurs.

Middle figure of Figure 3.4 shows a general case where $N > 1$. Right figure of Figure 3.4 incorporates the effect of CPU sleep. When there is nothing running on CPU, CPU goes to sleep mode. Let W be the wakeup time. Then peak at 0 moves to W. In reality, entire graph moves left by C, because consistent jitter of C can be removed.

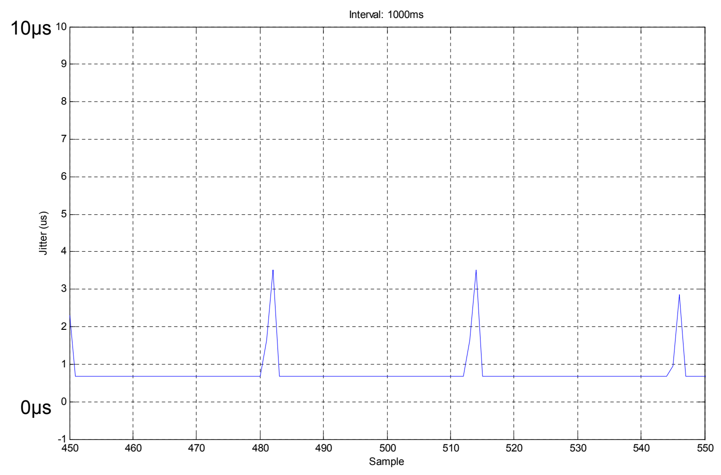


Figure 3.5: Time Series of Jitter at 1KHz

3.2.3 Removing Jitter

For high-frequency timer event, MicroTimer [1] is used instead of Timer [6] component. Timer component of TinyOS triggers at only up to 200Hz. MicroTimer supports only a single timer, but can trigger at up to several KHz. For light-weight EEPROM writing at high frequency, BufferedLog component is used.

From the jitter analysis in the previous subsection, we can see that the worst jitter is determined by the longest atomic section which can be running when the timer event occurs. This implies that we better give no chance for unnecessary components' atomic section to run on CPU. Therefore, we turn off every component except EEPROM during sampling. Then the assumption in the previous section holds, which says there can be only up to one atomic section running in when a timer event occurs. So jitter model is valid.

Figure 3.5, 3.6 and 3.7 show time series of jitter. Here is how it is measured. Hardware time is read when the timer event is switched onto the CPU. Since we know when sampling started and how many hardware ticks are equivalent to the interval, we

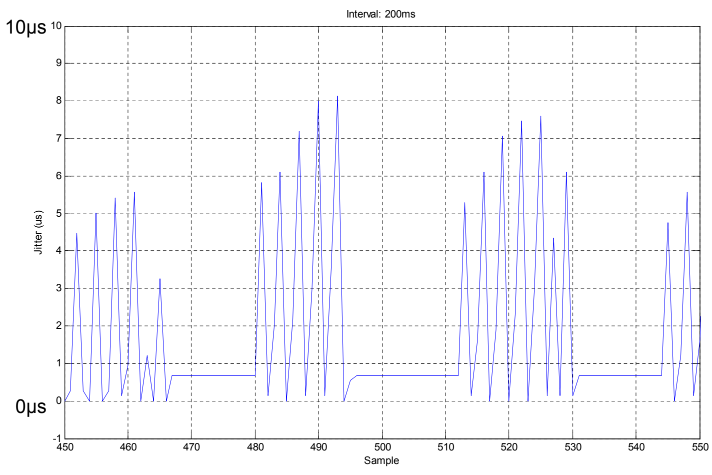


Figure 3.6: Time Series of Jitter at 5KHz

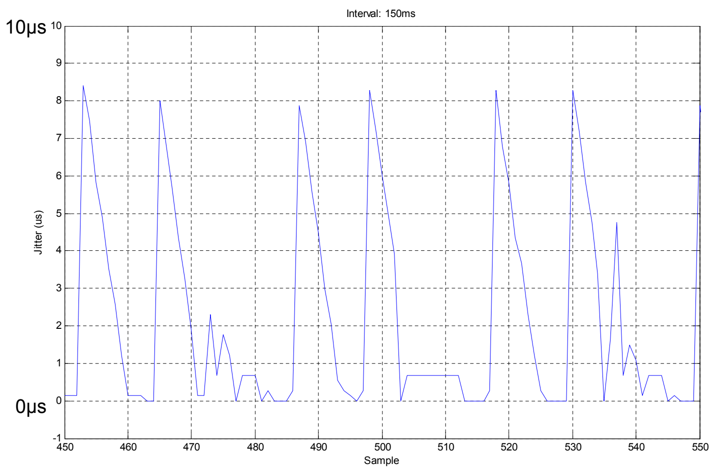


Figure 3.7: Time Series of Jitter at 6.67KHz

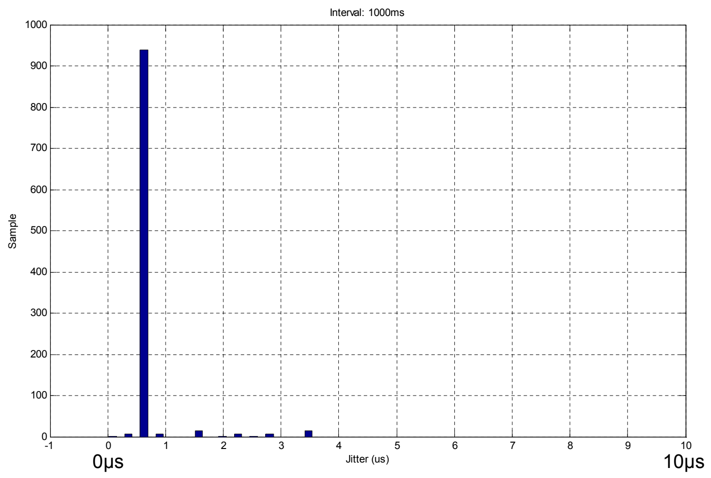


Figure 3.8: Histogram of Jitter at 1KHz

can calculate the jitter. There are two sections: plain section, spiky section, even though at 6.67KHz this separation is not clear. Let us define one epoch be a period of time to fill up a RAM buffer. These two sections constitute one epoch. During spiky period, the buffer is written to flash memory as a background task. At 1KHz, only a small portion of sampling is affected by flash memory write. At 6.67KHz, flash memory write affects too many sampling: most of sampling are affected by flash memory write. Looking at 5KHz case, even at 6.67KHz flash memory write should not affect that many sampling. However, the overhead of sampling itself seems to have some effect.

There is another interesting thing. At the plain section, there is a constant delay for every sampling. This delay is the wake up time of the CPU. When the CPU is idle, it enters a sleeping mode. And it takes 4 cycles to recover. Since there is a function call to record time, actually it takes 5 cycles here. Since the CPU runs at 8MHz, this wakeup time is equal to 625ns.

Figure 3.8, 3.9 and 3.10 show the distribution of jitter values (histogram). We can see a peak at 625ns, which is a wakeup time W. Except this peak, the frequency of jitter is largest

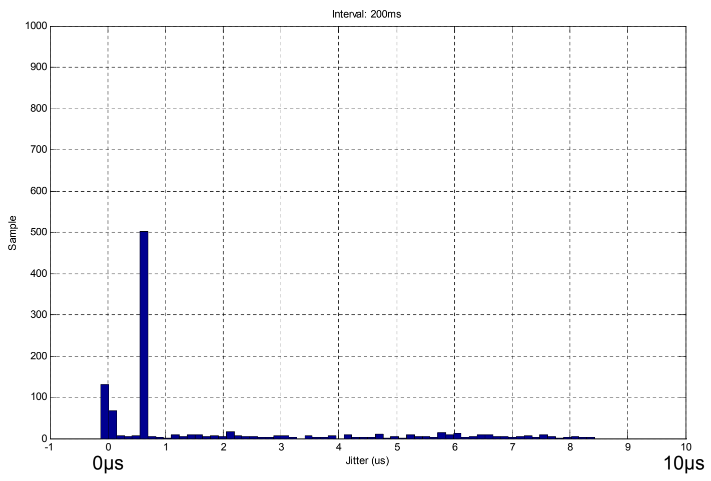


Figure 3.9: Histogram of Jitter at 5KHz

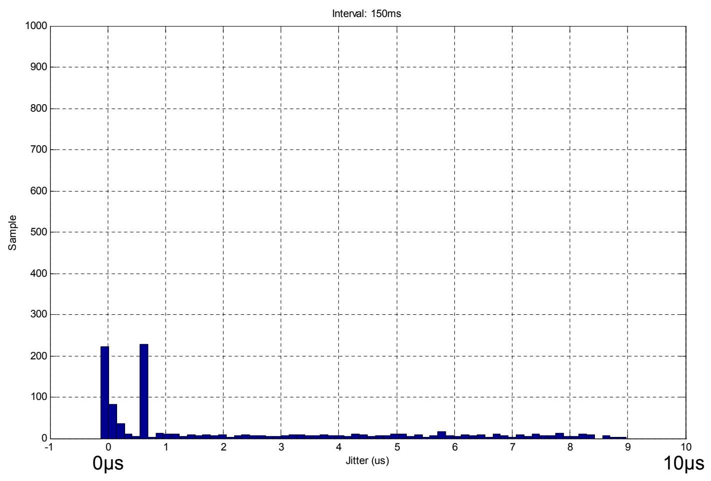


Figure 3.10: Histogram of Jitter at 6.67KHz

near 0s, and gradually decreases as jitter value increases. This result from real experiments matches quite well with the theoretical model. And jitter values are within 10μs.

3.3 Data Collection

Among the six challenges in Chapter 1, large-scale multi-hop network, reliable command dissemination and reliable data collection fall into data collection. Large-scale multi-hop network is provided by MintRout. Reliable command dissemination is achieved by retransmission. Here reliable data collection will be discussed.

Structural health monitoring and machine monitoring require all data from every node. Incomplete data set makes analysis extremely difficult or impossible. And many more applications need all the data.

It is desirable to pursue reliability at the minimal expense of other properties. Sacrificing channel capacity is not affordable. It is desirable for data collection to scale over multi-hop network. We also want bandwidth to remain high. In applications like structural health monitoring, sampling cycle is determined by data collection time, since sampling is quick, and collection takes most of time. And we like to minimize resource usage, because wireless sensor nodes are limited in computational power, memory space and energy

Here we introduce reliable data collection service having those properties: Straw (Scalable Thin and Rapid Amassment Without loss). Straw works on multi-hop routing layer providing good scalability. Complexity is drawn to a receiver (PC), and a sender (wireless node) is kept simple and light-weight. A round trip time and an interval between packets are optimized according to the sender node's depth in a routing tree, so high packet throughput is achieved without overwhelming the network blindly. Subsection 3.3.1 describes the high level protocol, Subsection 3.3.2 shows detailed implementation for optimization, and Subsection 3.3.3 will evaluate the performance and the efficiency of Straw.

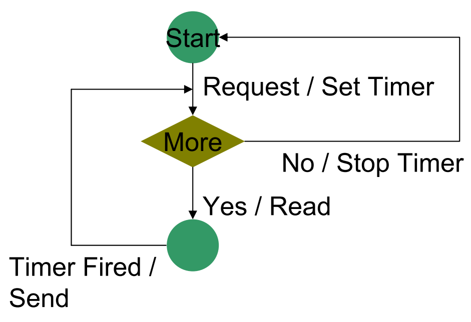


Figure 3.11: State Diagram of Sender

3.3.1 Protocol

At the high level view, the sender sends the entire bundle once, and the receiver asks for retransmission of missing bundle fragments.

Let us define terms first. A bundle is a unit of transfer, so Straw sends one bundle at a time reliably. A bundle is divided into bundle fragments. In Straw, one bundle fragment is one packet.

Straw uses end-to-end communication, so the receiver is always the PC. The sender is simple, and most of complexity is in the receiver, which has much more resources (computational power, memory space). The receiver initiates transfer and keeps states.

Sender

The sender responds to requests with simple sequences of operations. Figure 3.11 shows state diagram of the sender. The sender accepts three commands. First command is network information query. The sender replies with parent, depth, etc. The receiver optimizes parameters with them. Second command is transfer request. The sender sends the entire bundle once. Third command is random read. A request comes with a list of missing bundle fragment sequence numbers. Then, the sender reads those data with random

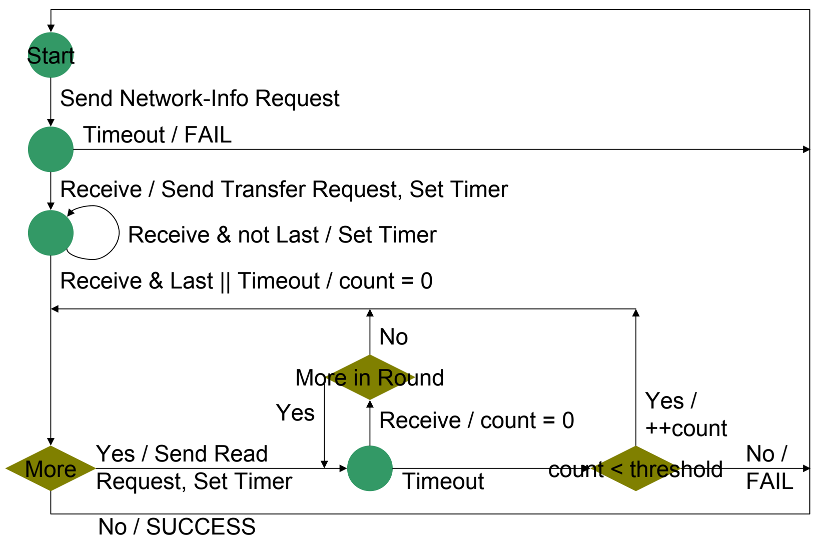


Figure 3.12: State Diagram of Receiver

access, and sends reply for each missing sequence number.

Receiver

The receiver is more complicated, as its state diagram shows in Figure 3.12. The receiver first asks for network information to adjust the round trip time and the interval between packets. Then the receiver requests data transfer. The entire bundle is transferred once. Then the receiver searches for missing bundle fragments, packs sequence numbers of them into one packet, sends it, and receives missing bundle fragments. This is one round. At the end of each round, the receiver finds missing sequence numbers again, and sends a request. This process continues until all missing bundle fragments are received.
The receiver has a timeout in each stage. The timeout for network information is straight forward. For data transfer, a timer is set for the estimated time of the last packet. For a request for missing bundle fragments, a timer is set to the last packet of that round. The

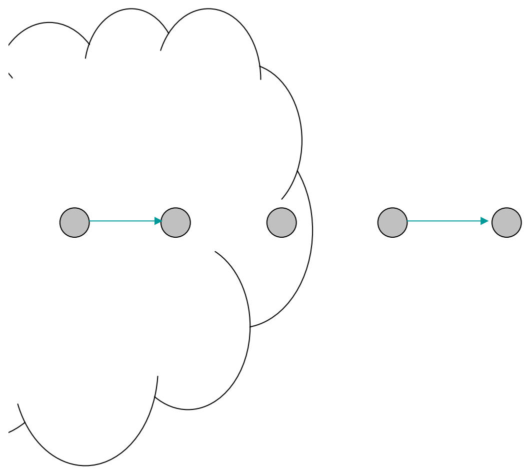


Figure 3.13: Range of Radio Interference

timeout for one round does not lead to a failure. After predefined number of consecutive timeouts, the receiver decides it as a failure.

3.3.2 Implementation

Straw is implementated in TinyOS targeting Mica2 mote. Many optimizations are put into implementation. Detailed discussion of them follows here.

Parameter Optimization with Network Information and Pipelining

The round trip time and the interval between packets are determined by the depth of the sender in the routing tree. The interval is

$$(Time\ through\ UART) + depth \times (Time\ through\ Radio)$$

Interference is assumed up to 2 hops. So as shown in Figure 3.13, when two senders are 3 hops away, it is assumed that both senders can transmit without much interference. So when the depth of sender is equal to or greater than 3, interval is

$$3 \times (Time\ through\ Radio)$$

to enable pipelining.

The depth in a routing tree can change over time. The sender continuously monitors its depth, and changes parameters dynamically.

The round trip time is

$$\{(Time\ through\ UART) + depth \times (Time\ through\ Radio)\} \times 2$$

In the implementation, 50% safety factor is added for times with possible uncertainty.

Collision Avoidance between Reply and Re-broadcast

There is a modification to reduce collisions. When a node at depth n receives command, it is re-broadcasted to nodes at depth n+1. Then nodes at depth n+1 will rebroadcast, and so on. Considering the assumption that nodes at 2 hops away will interfere, the reply from node at depth n needs to wait until nodes at depth n+2 finishes re-broadcasting. Therefore, a node needs to wait

$$3 \times (Time\ through\ Radio)$$

before sending reply. This factor is considered in waiting before sending reply and in estimating the round trip time.

Requesting Multiple Missing Bundle Fragments

The receiver asks for multiple missing bundle fragments in a single packet to reduce the overhead of NACK.

Rotating Buffers

Multiple buffers are used so that network transfer and memory read (either RAM or FLAHS, or something else with proper addressing) can overlap. And by having multiple buffers, the sending queue always has something to send. The number of buffers effectively determines maximum occupancy of Straw in the sending queue.

Simple Interface

Straw has very simple interface. Users need to call JAVA code at PC

$$int\ read(int\ dest,\ long\ start,\ long\ size,\ byte[]\ bffr)$$

and implement TinyOS code at nodes

$$event\ result\_t\ read(uint32\_t\ start,\ uint32\_t\ size,\ uint8\_t * \ bffr)$$

This gives an illusion of virtual tunneling.
Simplicity has tradeoffs. Efficiency in resource usage and generality decreases. However, many potential users are likely to be application developers who may not want to be entangled with buffer management for some more efficiency. Therefore, simple interface is selected in favor of an easy use.

Additional Features

For debugging, or reading data directly through UART like using a testbed, Straw supports the usage of UART instead of multi-hop routing.
Straw is flexible in bundle size from 1 byte to 1,310,710 bytes.
A manual, example codes can be found on website [2].

3.3.3 Evaluation

For evaluation, Broadcast [3] is used for a dissemination of command, and Route [4] is used as a routing layer. Figure 3.14 shows test data on Mica2 (433MHz radio) testbed in Soda Hall. 16 nodes are used in the test. The packet size was default 36 bytes. In this test, bandwidth was up to 576B/s for 1 hop case and up to 304B/s for 2 hop case.

To find out how much bandwidth is used out of the theoretical limit, a single hop case is studied further. Figure 3.15 shows how the maximum bandwidth is measured. As we decrease the interval between packets, success rate decreases also. By dividing success rate

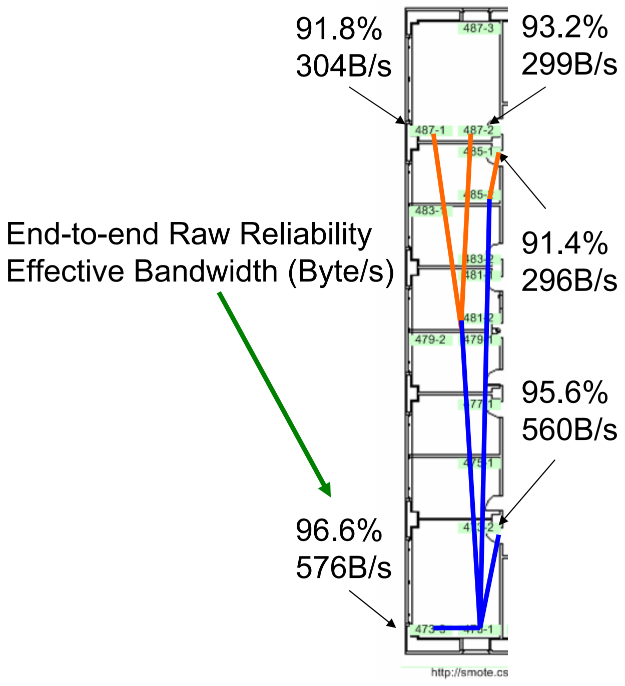


Figure 3.14: Test result on Mica2 testbed

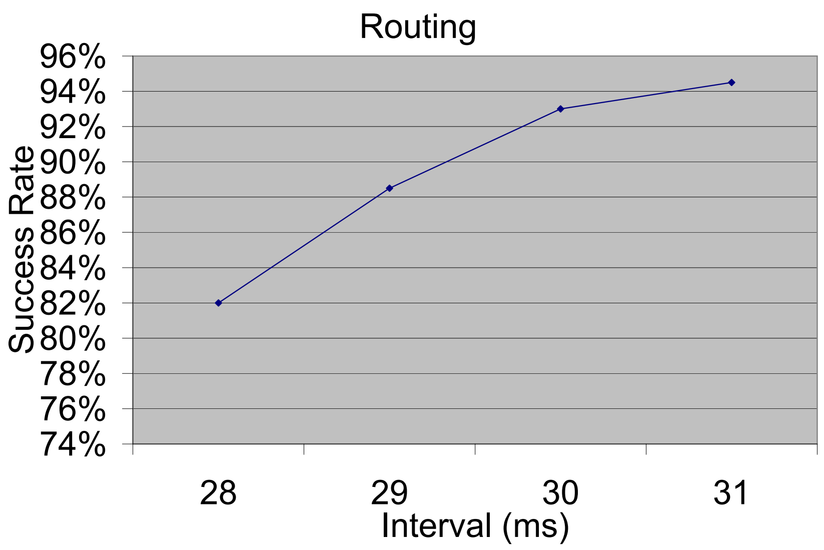


Figure 3.15: Example of Graph (Success Rate versus Packet Interval) used to measure maximum bandwidth

by the packet interval, effective bandwidth is obtained. Varying the packet interval, the maximum bandwidth is measured.

Bandwidth

Table 3.1 shows the bandwidth for UART, radio, 1 hop (radio and UART). The usable capacity is slightly over 60% of the hardware channel capacity. The theoretical capacity does not consider a preamble for MAC. Event handling overhead, MAC access

	Channel Capacity	Usable Capacity
UART	200.0pkts/s	120.4pkts/s
Radio	66.7pkts/s	42pkts/s
1 hop	50.0pkts/s	29.4pkts/s

Table 3.1: Maximum Channel Capacity and Measured Usable Capacity

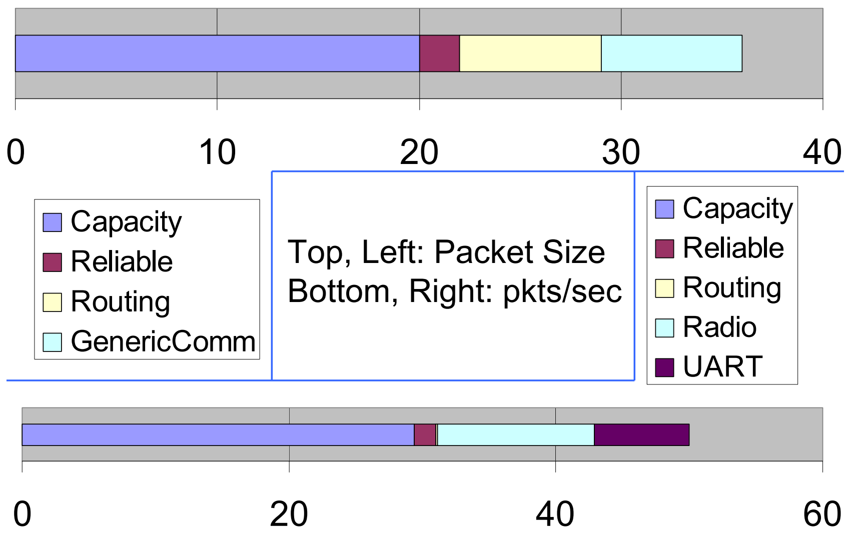


Figure 3.16: Payload and Packet Throughput

overhead, and preamble are some of reasons for 40% loss. Straw provides 29.4pkts/s which is 94.8% of the routing layers packet throughput.

Figure 3.16 shows loss in payload due to headers, and loss in packet throughput. We can see 44.4% of payload is used for headers. Packet throughput is shown as a pie again in Figure 3.17 for better analysis. 58.8% is usable out of the channel capacity. Figure 3.18 shows combined effect of those two on the bandwidth. Only 32.7% is usable above Straw.

To track where time is spent, Figure 3.19 shows packet time. The radio overhead (preamble, MAC) adds significant amount of overhead. And headers add much more burden. The overhead by protocols on upper layers are relatively small. Here we can find an opportunity for increasing the bandwidth. Even though we can't decrease the header size, we can increase the packet size so that relative header overhead decreases. And even if the packet size increases, the radio overhead will remain the same, and relative overhead of radio decreases.

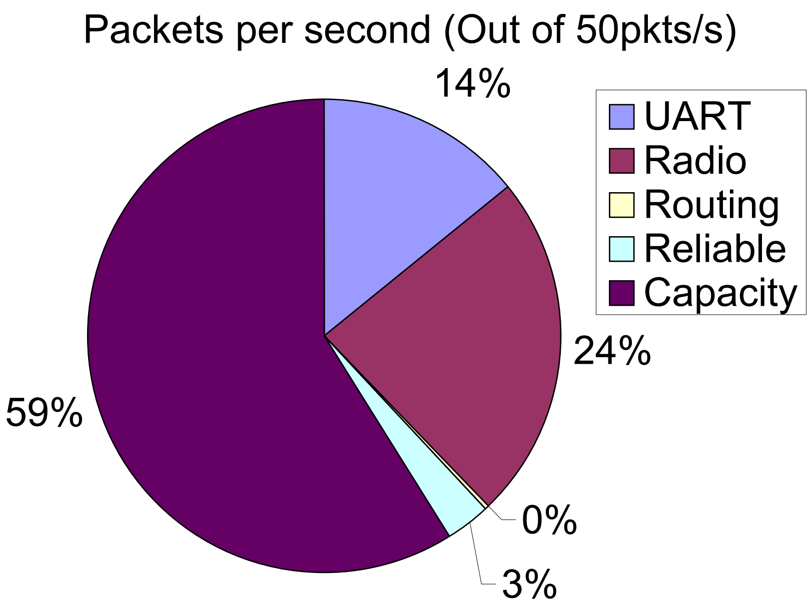


Figure 3.17: Packet Throughput in Pie Graph

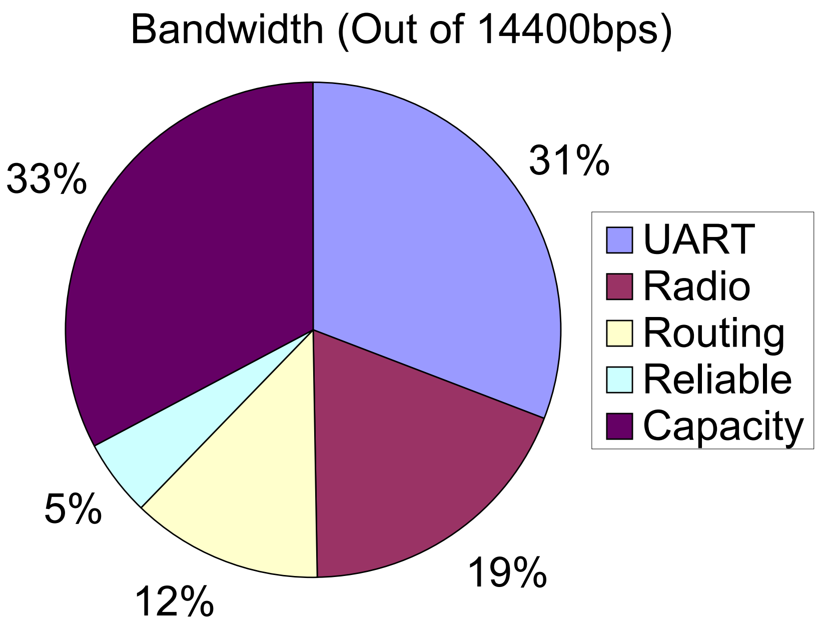


Figure 3.18: Effective Bandwidth. 33% is usable

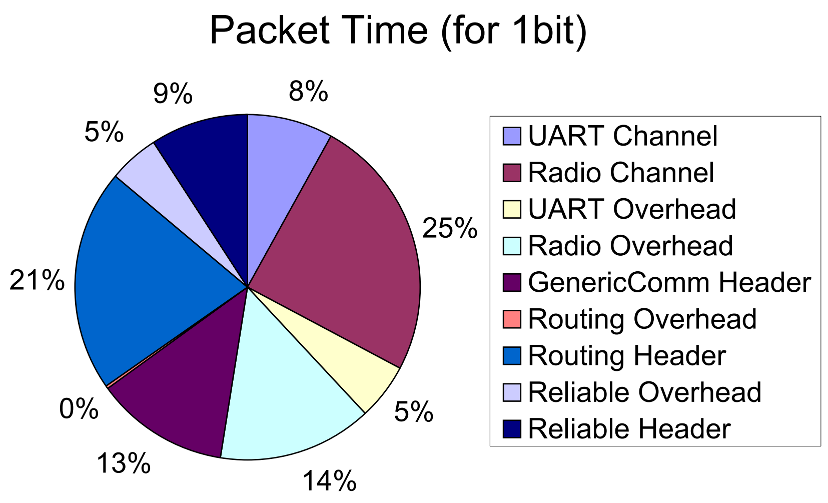


Figure 3.19: Graph showing where time is spent. Total time is 212.7μs

Larger Packet

A larger packet is tested. 36 byte packet is doubled to 72 bytes. The payload increases from 20B to 56B. Since the size of header is fixed, the payload increases by 2.8 times rather than 2 times. The packet throughput decreased from 29.4pkts/s to 20.9pkts/s. The radio overhead and the protocol overhead don't change much, so the packet throughput goes down only to 71%, not to 50%. 71% is slightly worse than the theoretical calculation 75%, which is obtained by doubling UART channel and Radio channel time in Figure 3.19 (additional overhead at radio and protocol will explain 4% decrease). This combination increases the bandwidth by 1.99 times (from 588B/s to 1172B/s).

When a loss rate is high, a larger packet means a higher loss rate. 1.99 is optimistic in a sense that the test environment has a high success rate (99.8%). However, since the payload is so small compared to a header with 36 byte packet, in many cases the benefit of a larger packet size would exceed the disadvantage by an increased loss rate.

3.3.4 Findings

Straw provides reliability, and achieves 94.8% packet throughput and 86.2% channel utilization on top of a routing layer in our controlled indoor testbed. And doubling packet size yields twice larger effective bandwidth, due to the small packet size.

Custody Transfer

Custody transfer is sending responsibility with data to the next hop. Therefore, the node at the next hop will make sure data get transfered reliably to the destination. When a larger packet is used, the importance of loss rate rises. With a larger packet, benefit of custody transfer becomes more apparent. Rather than going all the way between the sender and the receiver, local fix will be very efficient.

RAM Space

A larger packet is attractive for increasing the bandwidth. However, it also brings a new problem other than the effect of loss rate. 1 byte increase in packet size resulted in 33 bytes increase in RAM space with the test code, which means 33 packet buffers are used. When the packet size is double to 72 bytes, even basic services (time synchronization, broadcast, multi-hop routing, and reliable data collection) and a moderate application can use more than 4KB of RAM in mica2. Even the test program exceeded 4KB limit, so packet buffer size of routing layer had to be reduced from 16 to 12.

Figure 3.20 shows where packet buffers are used. First usage is as a buffer at end components. In TinyOS, packet space is provided by end components, so even when some component rarely sends packet, it still has to reserve packet space. Second usage is as a forward queue. There exists a mismatch between the incoming speed and the outgoing speed. Not to drop packets, a forwarding queue is needed. And the queue is managed by components requiring forwarding. And the size of buffer is related to the reliability: to provide higher reliability, the buffer size needs be larger. So to increase the reliability overall,

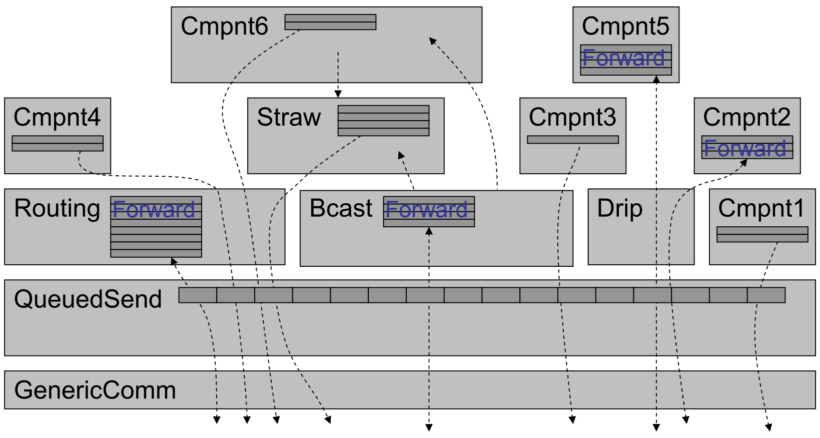


Figure 3.20: Usage of Packet Buffer

the size of forward queue needs be increased in each component with a forward queue.

Debate on Packet Buffer Pool

There is an alternative possibility for a packet buffer. Actual buffer space is provided by the lower layer at sending queue, and the upper layer keeps pointers only. Then the size of the sending queue determines reliability of every forwarding queue. A downside is that even though this solution is great in theory, the dynamic allocation of memory space very often leads to a disaster as we have seen in history (evolution from C++ to JAVA): it was the major source of bugs. Is controlled and limited sharing of packet buffer pool allowable or should it remain forbidden?

3.4 Structure Monitoring Toolkit

Structure Monitoring Toolkit (SMT) is a general toolkit for structure monitoring and machine monitoring. Here are highlights of SMT.

- **Modularization**: It is highly modularized, so subcomponents can be reused. As in Figure 3.1, command interpreter, sample & log, and read & send are independent components on TinyOS side. JAVA side has command interpreter and operation processor, each of which can be reused easily.
- **Multi-layer command**: There are two layers of command: operation, micro-operation. Operation is a composition of micro-operations. Command-line operation is decomposed into micro-operations which motes understands. So it is easy to add or modify operations without changing program on mote side.
- **Stateless mote**: Since a mote executes simple micro-operations, a mote is almost stateless, so it is robust.
- **Meta-data management**: Meta data describing data is stored separately. So description and interpretation of data is clean and easy to modify.
- **Integrity**: Mote keeps integrity state at reset, erase and writing data. This guarantees partial imperfect data isn't read accidentally.
- **Useful tools**: There are many useful tools for collecting information about time synchronization, network, and sampling jitter.

Command list is in Appendix B.

Chapter 4

Data Analysis

4.1 Signal Processing

As an analog signal processing low-pass filter is used, which filters high frequency noise. However as shown in Figure 4.1, a low-pass filter is not perfect, and there exists some leftover signal above threshold frequency. Therefore even if a low-pass filter is used, sampling frequency at ADC should be higher than threshold frequency of low-pass filter. Moreover by the Nyquist theorem, to avoid aliasing, sampling rate should be at least twice of signals frequency. For accelerometer board, low-pass filter with threshold frequency 25Hz is used. Then ADC should sample at frequency much higher than 50Hz.
As a digital signal processing, signals are oversampled and averaged. If noise follows Gaussian distribution, by averaging N numbers, noise decreases by a factor of $\sqrt{N}$. This multiplies sampling frequency by a factor of N. We used $N = 5$ for a footbridge deployment.

4.2 System Identification

System identification is identifying the model of target system. By matching input to system and output from system, we can construct a mathematical system model. The usual process is fitting a general Box-Jenkins multi-input multi-output model to sampled

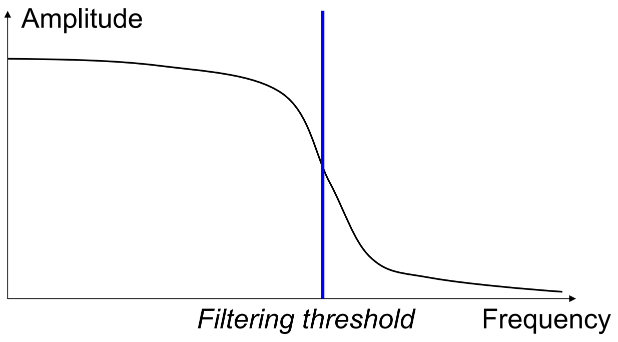


Figure 4.1: Imperfect Low-pass Filter

data. And natural frequencies, damping ratios and mode shape are then estimated using the estimated Box-Jenkins model. We will not cover the system identification further. More detail on the system identification can be found in [20].

A change in modal properties is a result of a change in structural stiffness (assuming that the masses do not change significantly). Deterministic and probabilistic methods have been developed to detect these changes and translate them to a structural damage (by using substructures for example).

Chapter 5

Results from Footbridge Deployment

The final target of the project is the Golden Gate Bridge. Before going there, we used a footbridge as a testbed. We tested radio and network in a bridge environment. And we actually measured the vibration of the bridge, and analyzed the collected data.
The Footbridge from City of Berkeley to Berkeley Marina spans over I-80. It is 260ft long and 16ft wide suspension bridge hang by steel arch as seen in Figure 5.1.

5.1 First Deployment

At the first deployment, two nodes are deployed. Figure 5.2 depicts the location of the nodes: one at mid-span, the other at quarter-span. Both nodes are close to the base station that they can directly communicate with the base station. Eigen Realization Algorithm was used to estimate the modal properties of the bridge (frequencies, damping ratios and mode shapes at those two locations). Figure 5.3 shows the first vertical mode of vibration. And Figure 5.4 shows the second vertical mode of vibration. Figure 5.5 and Figure 5.6 shows horizontal mode of vibration, first mode and second mode, respectively.

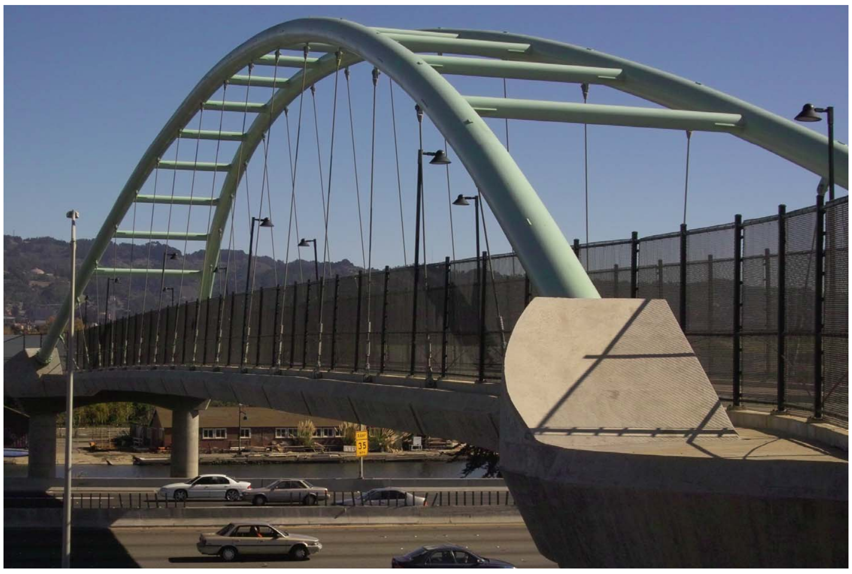


Figure 5.1: Footbridge Over I-80

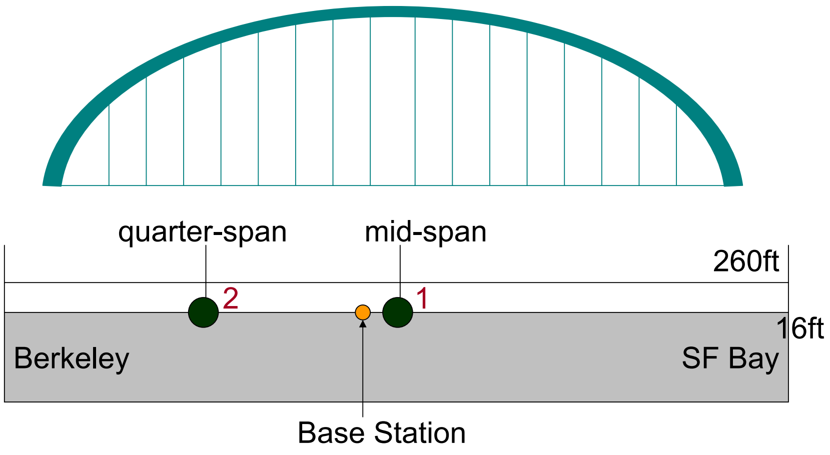
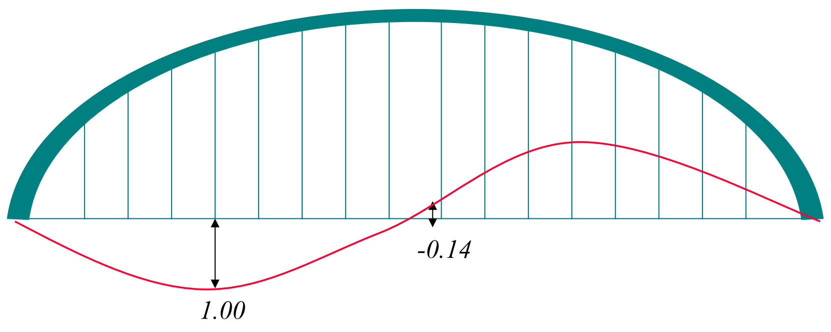
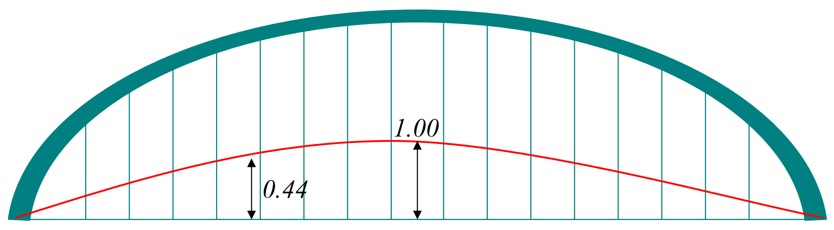


Figure 5.2: Location of Nodes at the First Deployment



First Vertical Mode of Vibration
Frequency: 1.35 Hz
Damping Ratio: 5.5%

Figure 5.3: First Vertical Mode



Second Vertical Mode of Vibration
Frequency: 1.79 Hz
Damping Ratio: 2%

Figure 5.4: Second Vertical Mode

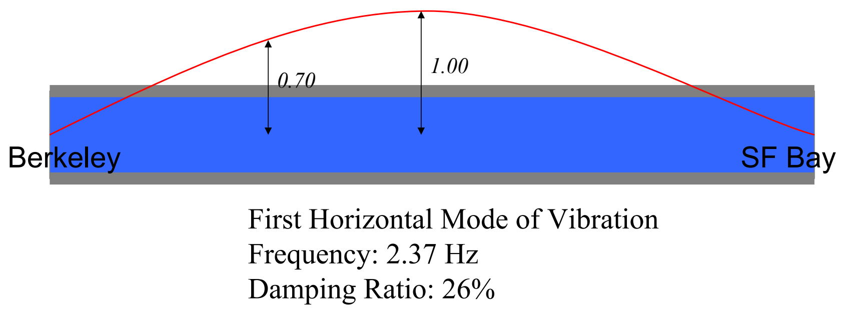


Figure 5.5: First Horizontal Mode

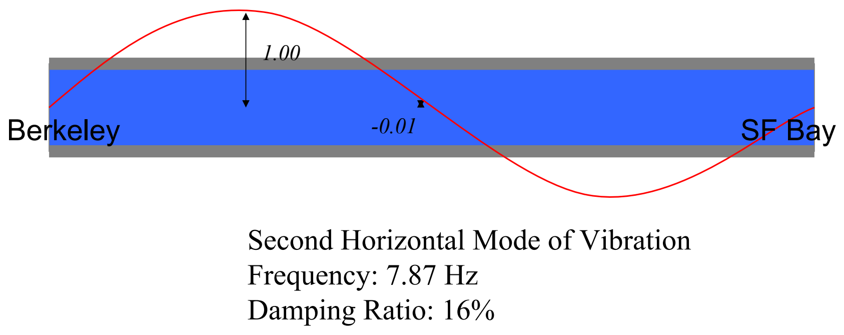


Figure 5.6: Second Horizontal Mode

		1st mode	2nd mode	3rd mode	4th mode
Vertical	Frequency (Hz)	1.35	1.79	11.47	13.75
	Damping Ratio	0.055	0.02	0.043	0.08
Horizontal	Frequency (Hz)	2.37	7.87	11.91	14.59
	Damping Ratio	0.26	0.16	0.123	0.092

Table 5.1: Maximum Channel Capacity and Measured Usable Capacity

Table 5.1 shows a summary of the vibrations. This can be thought as a part of a signature of the bridge. When the bridge suffered structure change, numbers in this table can change. However, by how much these numbers are affected by the change of the structure, and how much change in these numbers indicates hazards of the structure, and how many modes needs be looked at, remain as a future work.

5.2 Second Deployment

We went to bridge for the second deployment with improved hardware and software. We had the following objectives.

- Achieve higher spatial resolution to get a denser look at the vibration
- Measure torsion
- Evaluate the sensitivity of different algorithms to measurement/hardware noise
- Evaluate problems arising from deployment of a larger network

Figure 5.7 shows the location of motes. 13 motes are deployed. 10 of them were used to measure vibration of the bridge, 3 of them were used to see calibration accuracy and variation of boards. The deployment was successful. Multi-hop network was formed, and reliable communication worked. And all nodes sampled data synchronously. Figure 5.8,

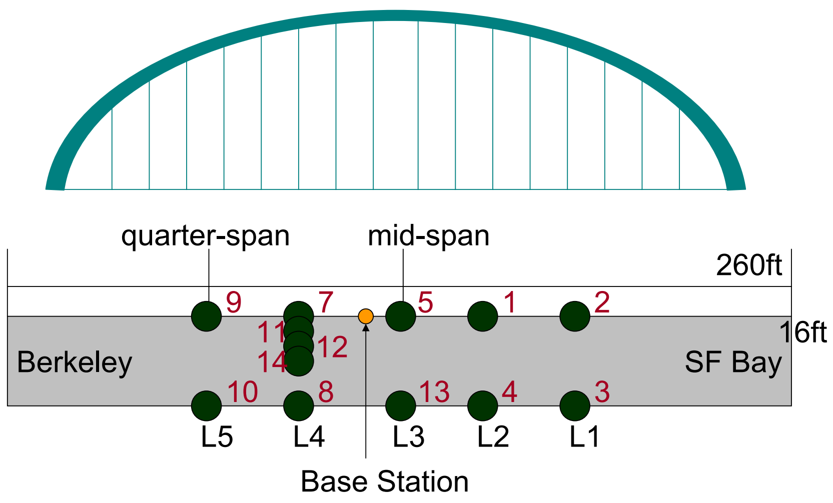


Figure 5.7: Location of Nodes at the Second Deployment

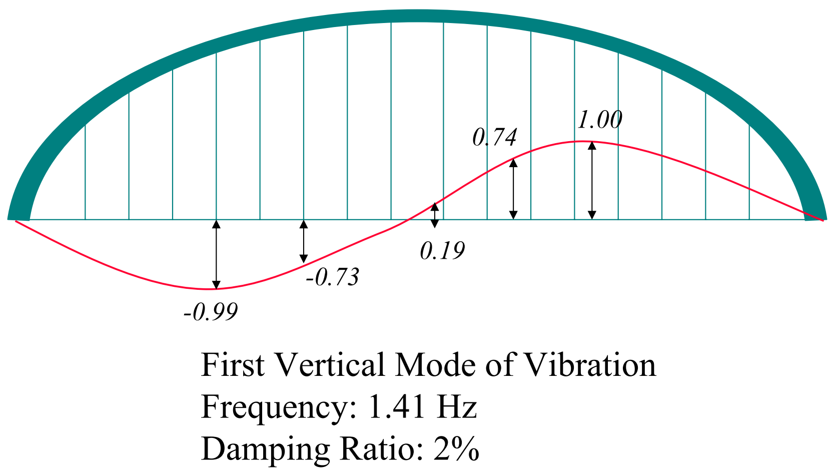


Figure 5.8: First Vertical Mode at the Second Deployment

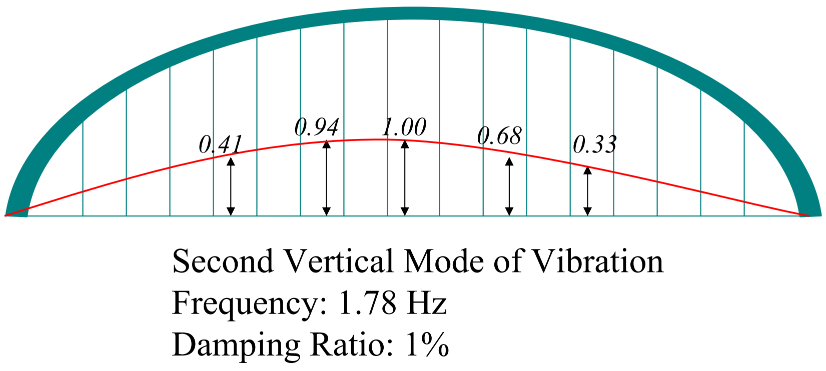


Figure 5.9: Second Vertical Mode at the Second Deployment

5.9 show the first and second vertical mode of vibration. The result coincides with that of the first deployment. Another interesting point is that vertical vibration is stronger than horizontal vibration. The reason can be found at the structure of the bridge as shown in Figure 5.10. Under the span, beams are supporting span. This gives rigidity to horizontal direction. Therefore, horizontal vibration is weak.

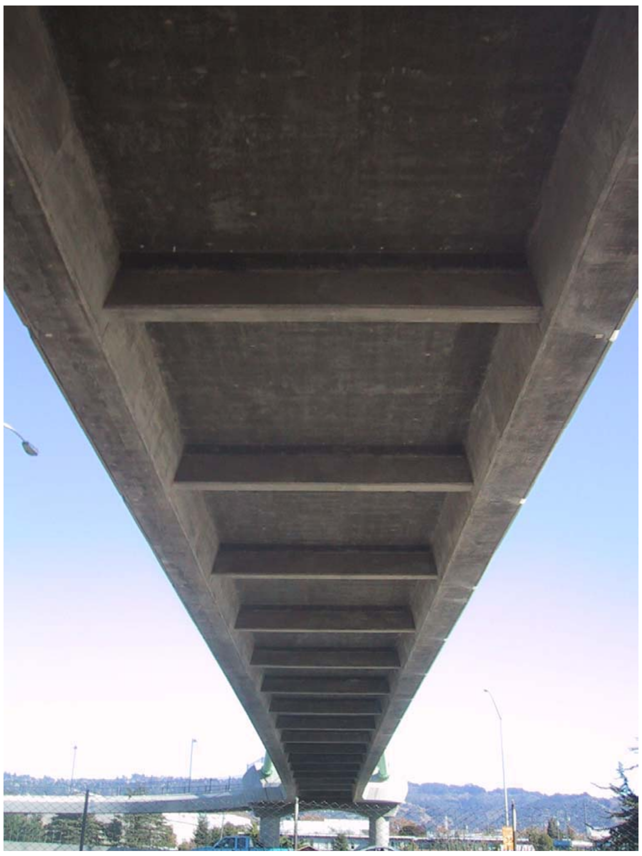


Figure 5.10: Beams Supporting Span

Chapter 6

Related Work

Habitat monitoring is a leading application of wireless sensor network. And it is an example application with low duty cycle. ZebraNet[18] uses PDA-level device with 802.11b wireless network. Great Duck Island[26] uses Berkeley mote, and watch petrels without disturbing them at low cost.
For structure monitoring, there are researches using conventional wired sensors. GPS was used combined with wired data collection[23, 14], however at a high cost.
There are approaches using WSN for SHM [13, 24, 19, 15]. MEMS accelerometers are used to sample vibration. RF radio is used for the wireless communication. And microcontroller drives sensors and radios. However, they all use custom radio, and can not scale to multiple nodes over multi-hop network. [17] uses Mica2 and TinyOS. It supports multi-hop network, however there is no provision for the time synchronization, which is necessary for time-related sampling of the entire structure.

[21, 28] has time synchronization and reliable multi-hop communication. However, there is no study about the required fidelity of data. Data should meet requirements and challenges in Chapter 1 for SHM.
On hardware side, accuracy of hardware is not good enough to capture 500μG signal. Study

for calibration is absent. To capture the required signal, a new board would be needed, and whole body of calibration should be studied.

On software side, there is no study on sampling jitter. Time synchronization accuracy and reliability of communication are not good enough to capture meaningful signal from the structure. When higher quality of data is needed, significant amount of software needs be improved and analyzed.

So even though these works are good examples showing the possibility of using WSN in SHM, produced data is not of adequate quality to be used for meaningful analysis.

Chapter 7

Conclusion

Lessons from this project will be presented first; contributions and conclusions will follow.

The accuracy of the accelerometer, installation tilt error, effect of temperature variation, sampling jitter, and reliability of data collection, all limit the fidelity of data. And the worst parts of them determines the quality of data. Even if the accelerometer is very accurate, the quality of sample will depend on the accuracy of calibration, if temperature calibration is poor. As we increase the requirements for sample quality, more parts fall below the threshold, and more work is needed to make them satisfy the required level. So when a new application is considered, the requirement for data quality needs be investigated for estimating the required accuracy and complexity of the system.

Another difficulty was that some problems do not fall within computer science or civil engineering. For example, antenna and packaging are critical problems. But antenna belongs to a field of communication, so we needed to get help from experts in that field. Packaging was in the same situation: mechanical engineering. And real implementation and deployment required some engineering effort: soldering wires and drilling holes.

One subtle issue came from the nature of interdisciplinary project. Even though it is not always true, people in computer science tend to like to deploy and iterate, while people

in civil engineering prefer thoroughly perfecting system first. In computer science, except in the case of mission-critical software like one in space shuttle, we can quickly debug problems found. In many cases, it is more reasonable and economical to deploy before catching every single bug, which would be impossible for a large system. So the mindset is inclined to deploying, testing, debugging, and iterating. In contrast, in civil engineering, we can not debug and patch collapsed bridge and lost lives. Therefore, it is very compelling to make sure the system does not have any problem before deployment.
However, we could appreciate the approach of each side. Quick deployment and iteration can discover that some problems are not as critical as we assume, or more serious than we think. Careful design and test can prevent costly change at later stage. The questions would become what is good tradeoff.

This study makes three main contributions to wireless sensor networks.

- First contribution is research on high frequency sampling. Supporting components for high frequency sampling are used and tested. Limitation of system is examined. Condition-based machine monitoring [7], and sound detection [25] are examples of applications requiring high frequency sampling. They will benefit from this study.

- Second one is reliable data collection component. Reliable Data Collection (Straw) is scalable, light-weight on mote, reliable, and achieves very high packet throughput and channel utilization. It is an independent separate component, which is easy to plug in. There exist huge categories of applications (including two examples above: condition-based machine monitoring and sound detection) which need every data.

- Third one is integration of diverse services, and driving them to an extreme. Multi-hop routing, reliable dissemination, reliable data collection, time synchronization, and high frequency sampling, are all integrated in a single application. And they are operated to an extreme degree, putting them under significant stress. This found that current network abstraction requires many duplicate packet buffers consuming large amount of RAM space. And we found bugs which showed up only under extreme network stress.

Deployment on the footbridge showed the system captures signal successfully, and we successfully analyzed the bridge with collected data.

Through this work, WSN supports the opposite extreme of its usual application so far: sampling high fidelity data at high frequency, and collecting large amount of data reliably.

Bibliography

[1] http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/apps/HighFrequencySampling/MicroTimerM.nc.

[2] http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/contrib/GGB.

[3] http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/tos/lib/Broadcast.

[4] http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/tos/lib/Route.

[5] http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/tos/system/BufferedLog.nc.

[6] http://cvs.sourceforge.net/viewcvs.py/tinyos/tinyos-1.x/tos/system/TimerC.nc.

[7] http://www.intel.com/research/vert_manuf_condmaint.htm.

[8] http://www.ni.com/labview.

[9] http://www.superpass.com/SPPG24BD.html.

[10] http://www.tadiranbat.com/prodpdf/viewpdf.php?datasheet=TL-5930.pdf.

[11] http://www.tinyos.net/scoop/special/hardware.

[12] A. M. Abdel-Ghaffar. Ambient vibration studies of golden gate bridge. *Journal of Engineering Mechanics*, 111(4):483–499, April 1985.

[13] J. M. Caicedo, J. Marulanda, P. Thomson, and S. J. Dyke. Monitoring of bridges to detect changes in structural health. *the Proceedings of the 2001 American Control Conference, Arlington, Virginia, June 2527, 2001.*

[14] P. Cheng, W. J. Shi, and W. Zheng. Large structure health dynamic monitoring using gps technology.

[15] J. M. Engel, L. Zhao, Z. Fan, J. Chen, and C. Liu. Smart brick - a low cost, modular wireless sensor for civil structure monitoring. *International Conference on Computing, Communications and Control Technologies (CCCT 2004), Austin, TX USA, August 14-17, 2004.*

[16] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for network sensors. *ASPLOS 2000, Cambridge, MA, November 2000.*

[17] B. S. Jr., M. Ruiz-Sandoval, and N. Kurata. Smart sensing technology: Opportunities and challenges. *Journal of Structural Control and Health Monitoring, in press, 2004.*

[18] P. Juang, H. Oki, Y. Wang, M. Martonosi, L.-S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *the Proceedings of ASPLOS-X, San Jose, October 2002.*

[19] M. Krger and C. U. Grosse. Structural health monitoring with wireless sensor networks. *Otto-Graf-Journal*, 15:77–90, 2004.

[20] L. Ljung. *System Identification. Theory for the user.* Prentice-hall, 1987.

[21] J. P. Lynch, A. Sundararajan, K. H. Law, A. S. Kiremidjian, E. Carryer, H. Sohn, and C. Farrar. Field validation of a wireless structural monitoring system on the alamosa

canyon bridge. *SPIE 10th Annual International Symposium on Smart Structures and Materials, San Diego, CA, USA, March 2-6, 2003.*

[22] M. Marti, B. Kusy, G. Simon, and kos Ldeczi. The flooding time synchronization protocol. *the Proceedings of ACM Second International Conference on Embedded Networked Sensor Systems (SenSys 04), pp. 39-49, Baltimore, MD, November 3, 2004.*

[23] C. Ogaja, C. Rizos, J. Wang, and J. Brownjohn. Toward the implementation of on-line structural monitoring using rtk-gps and analysis of results using the wavelet transform.

[24] P. Qiang, G. Xun, and Z. Chang-you. A wireless structural health monitoring system in civil engineering. *The Third International Conference on Earthquake Engineering (3ICEE), Nanjing, China, October 18-20, 2004.*

[25] G. Simon, M. Marti, kos Ldeczi, G. Balogh, B. Kusy, A. Ndas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. *the Proceedings of ACM Second International Conference on Embedded Networked Sensor Systems (SenSys 04), pp. 39-49, Baltimore, MD, November 3, 2004.*

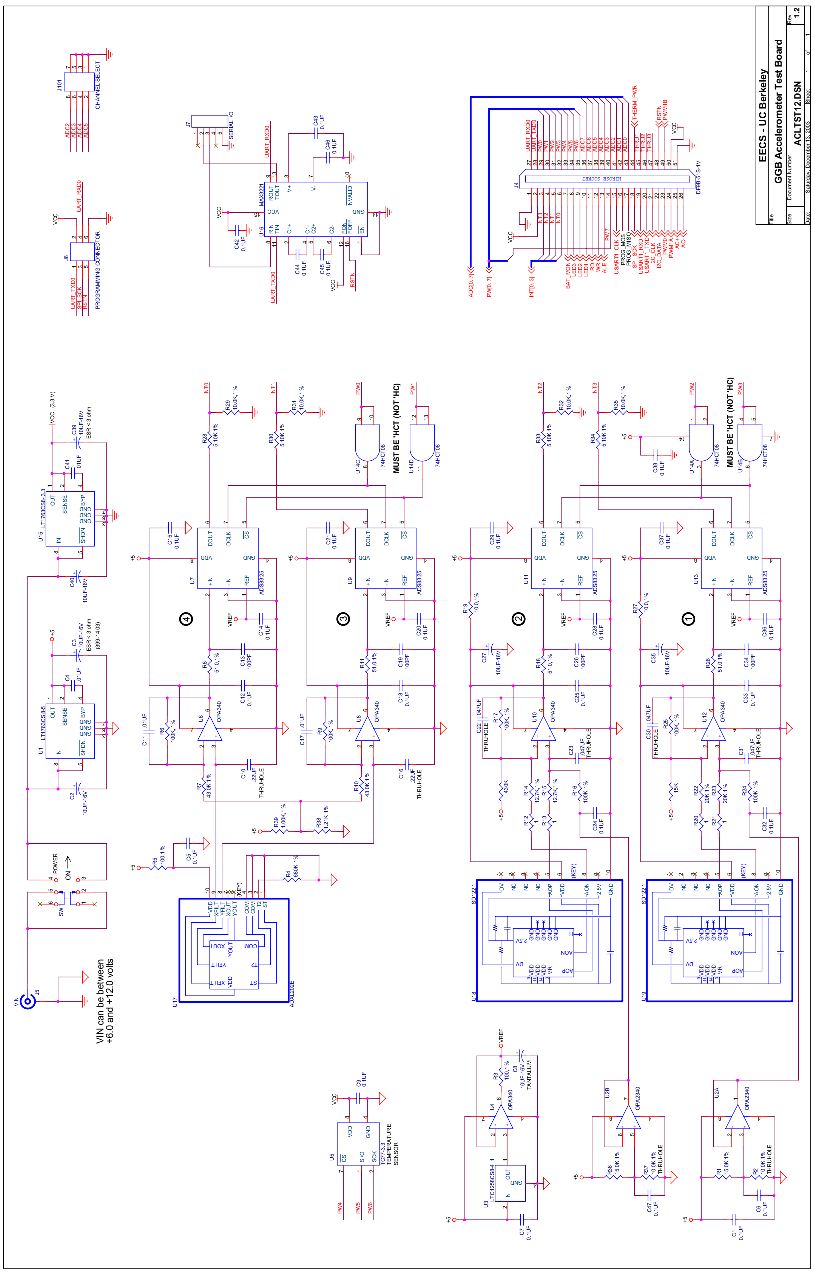
[26] R. Szewczyk, A. Mainwaring, J. Polastre, and D. Culler. An analysis of a large scale habitat monitoring application. *the Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys), November 3-5, 2004.*

[27] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multhop routing in sensor networks. *SenSys 2003 Los Angeles, California.*

[28] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. *the Proceedings of the ACM Conference on Embedded Networked Sensor Systems, November 2004.*

Appendix A

Schematics of Accelerometer Board


EECS - UC Berkeley
Title
GGB Accelerometer Test Board
Size
Document Number
ACLTST12.DSN
Rev
1.2
Date: Saturday, December 13, 2003
Sheet 1 of 1
VIN can be between +6.0 and +12.0 volts
MUST BE 'HCT (NOT 'HC)
PROGRAMMING CONNECTOR
CHANNEL SELECT
TEMPERATURE SENSOR

Appendix B

Structure Monitoring Toolkit (SMT) Commands

```
Usage: java net.tinyos.SMT.DataCenter <command> <arguments> [options] where
    <command> <arguments> [options] can be one of the following:

ledOn [-dest dest] [-verbose]
ledOff [-dest dest] [-verbose]
pingNode <dest> [-toUART] [-verbose]
nodeList [-toUART] [-verbose]
eraseFlash [-dest dest] [-verbose]
startSensing <nSamples> <intrv> [-dest dest] [-chnlSelect chnlSelect]
    [-samplesToAvg samplesToAvg] [-nm nm] [-verbose]
eraseStart <nSamples> <intrv> [-dest dest] [-chnlSelect chnlSelect]
    [-samplesToAvg samplesToAvg] [-nm nm] [-verbose]
readProfile [-dest dest] [-toUART] [-verbose]
readData [-dest dest] [-toUART] [-verbose]
randomRead <dest> <chnlNo> <sampleNo> [-toUART] [-verbose]
```

```
timesyncInfo <dest> [-toUART] [-verbose]

networkInfo <dest> [-toUART] [-verbose]

forDebug [-dest dest] [-toUART] [-verbose]

resetBcSeqNo [-verbose]

reset [-dest dest] [-verbose]

help [-verbose]
```