A PROGRAMMABLE WIRELESS SENSING SYSTEM FOR STRUCTURAL MONITORING

Jeongyeup Paek, Omprakash Gnawali, Ki-Young Jang, Daniel Nishimura, Ramesh Govindan
Computer Science Department, University of Southern California
jpaek@usc.edu, gnawali@usc.edu, kjang@usc.edu, dan@enl.usc.edu, ramesh@usc.edu

John Caffrey, Mazen Wahbeh and Sami Masri
Civil Engineering Department, University of Southern California
jcaffrey@usc.edu, mwahbeh@worldnet.att.net, masri@usc.edu

Abstract

Recent work has examined the design of wireless sensor network (WSN) systems for structural health monitoring (SHM). Wireless sensors enable dense monitoring of large physical structures and promise enormous ease and flexibility of deployment of instrumentation, as well as low maintenance and deployment costs. However, programming sensing applications on a network of wireless sensors remains a difficult and time-consuming endeavor. This is due in part to the complexity of such systems. Their limited battery resources, and the highly variable performance of wireless communication in different environments represent significant constraints that, if each application developer were forced to deal with, can significantly increase the time to develop robust applications. We have been developing a networked software system called TENET that simplifies the programming of wireless sensor actuator systems. A TENET system is a two-tier networked system consisting of two classes of nodes: a higher-tier with several nodes containing 32-bit processors and IEEE 802.11b radios, and a lower-tier comprising battery-operated sensor nodes with less-capable processors, low-power radios. Our TENET software runs application code on the higher-tier nodes, and provides a generic interface for tasking sensors and actuators. This separation of functionality simplifies application development greatly, since developers can reuse networking and sensor data extraction code, thereby reducing application development time. We will report on the development of and experiences with structural data acquisition application for a long-span suspension bridge using TENET. We will report on our experiences in deploying a two-tier network of wireless sensors on the bridge. We will report on the performance of the TENET system in this setting as well.

Introduction

Structural Health Monitoring (SHM) focuses on developing technologies and systems that assess integrity of structures such as buildings, bridges, Earp-space structures and off-shore oil rigs Doebling *et al.* (1996). Most existing SHM implementations use wired data acquisition systems to collect structural vibration data from various locations in the structure for analysis. More recently, with the advent of low-cost platforms for wireless sensing, wireless structural monitoring systems become feasible. Such wireless structural monitoring systems promise ease and flexibility of deployment in addition to low maintenance and deployment costs.

Much recent work in wireless structural monitoring has focused on the development of hardware and node platforms (Lynch & Loh, 2005). little attention has been devoted to the *software* for such systems, yet a standardized software system that provides the right level of programming abstraction can greatly advance wireless structural sensing. Today, programming sensing applications on a network of wireless sensors remains a difficult and time-consuming endeavor. This is due in part to the complexity of such systems. Their limited battery resources, and the highly variable performance of wireless communication in different environments represent significant constraints that, if each application developer were forced to deal with, can significantly increase the time to develop robust applications.

In this paper, we report on the design of a generic sensing software called TENET. TENET is designed for a two-tier network of sensors and actuators: a lower tier consisting of low-power sensing nodes which we generically call motes and which enable flexible deployment of dense instrumentation, and an upper

tier containing fewer, relatively less constrained, 32-bit nodes with higher-bandwidth radios, which we call masters. In TENET, applications run on one or more master nodes. They *task* motes to sense and locally process data. Conceptually, a task is a small program written in a constrained language. The results of tasks are delivered by the TENET system to the application program. This program can then fuse the returned results, and re-task the motes or trigger other sensing modalities. More than one application can run concurrently on a TENET.

We have designed and implemented TENET and have implemented several applications on it (Gnawali *et al.*, 2006). In this paper, we report on the performance of a wireless sensor network running TENET on a long-span suspension bridge. We instrumented 600 ft of the suspension bridge using a tiered wireless network consisting of five master nodes and 20 motes, and collected data for 24 hours. We report on our experiences with the deployment, and provide some preliminary data about the structural characteristics of the bridge.

The TENET System Overview

TENET is a two-tier networked system consisting of two classes of nodes: a higher-tier with nodes (called Master) containing 32-bit processors and IEEE 802.11x radios, and a lower-tier comprising battery-operated sensor nodes with less-capable processors, and low-power radios such as IEEE 802.15.4. Our TENET software runs application code on the higher-tier nodes, and provides a generic interface for tasking sensors and actuators. This separation of functionality simplifies application development greatly, since developers can reuse networking and sensor data extraction code, thereby reducing application development time.

Our architecture is based on the assumption that future large-scale sensor networks will be *tiered*. Since motes have a small form-factor and are untethered, they enable flexible, low-cost, dense deployment of sensors; masters, which have significantly higher communication capacity and radio range, enable larger spatial coverage and, more important, *network capacity scaling*. Because master nodes have higher communication capacity, tiered networks scale better than similarly-sized flat networks of motes.

A TENET system is based on a design principle that clearly identifies the role of the masters and the motes. In TENET, any and all communication takes the form of a task and a response to a task. Only masters are able task a mote, and the motes are only allowed to send task responses back to the masters. Master send tasks to the motes and receive the response back from the motes. Motes provide a limited library of generic functionality, such as timers, sensors, thresholding, data compression, and other forms of simple signal processing. Each task activates a simple subset of this functionality.

Three important advantages arise from these design principles. First, applications execute on the master tier where application programmers can use familiar programming interfaces since that tier is relatively less constrained. Second, the mote tier *networking functionality is generic*, since TENET's networking subsystem merely needs to robustly disseminate task descriptions to motes, and reliably return results to masters. This enables significant code re-use across applications. Finally, mote functionality is limited to executing tasks and returning responses, enabling energy-efficient operation.

Tasks and Task Library

In the TENET task library, tasks are composed of *tasklets*. Each tasklet may be thought of as a service to be carried out as part of the task. The tasklets in the TENET task library provide the building blocks for a wide array of data acquisition, processing, filtering, measurement, classification, diagnostic, and management tasks.

For example, to construct a task that samples a particular ADC channel 0 every 1000 ms and transmit ten samples at a time, with the tag TEMPERATURE, to its master, we write:

```
Sample(500ms, 10, REPEAT, 1, ADC0, TEMPERATURE) -> SendPtr()
```

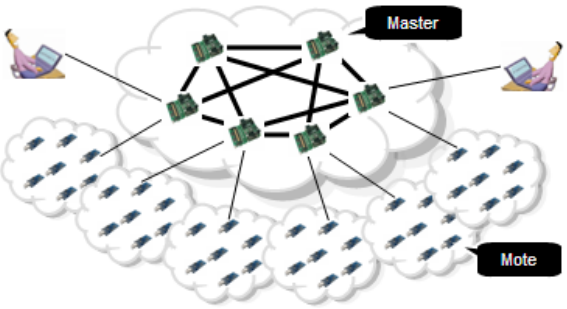

Figure 1: A tiered embedded network

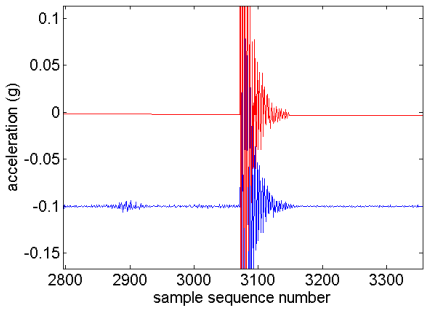

Figure 2: vibration Sensing with Onset-Detector

This provides the most basic sampling and transmission support one might expect from a mote. It periodically collects a single sensor value and delivers data using multi-hop transport. If the application programmer wishes to put a timestamp on every packet, and send the packet only if the the sample value is above 45, then the task description will be in the following form:

```
Sample (1000ms, 10, REPEAT, 1, ADC0, TEMPERATURE)
        -> StampTime(B, LOCAL)
        -> ClassifyAmplitude(45, 3, A, ABOVE)
        -> SendPtr()
```

This linear form of task construction are flexible, general, high-level, and efficient enough to support easy programming and tasking from masters.

How can a TENET application make use of this tasking library? Shown below is the code for *a complete application* which collects vibration data from three axes at sampling frequency of 20Hz. This application is exactly what we have used for our bridge deployment. Here, `SampleMDA400` is the tasklet that controls a specific vibration sensorboard, and `SendStr()` invokes our stream transport (described later).

```
int main()
{
    task_string = "SampleMda400(50000,0,X,Y,Z,TIME,7,15)->SendStr()"; /* task description */
    task_packet = construct_task(task_string);                        /* construct a task packet */
    task_id = send_task(task_packet);                                 /* send the task */
    while (1) {
        packet = receive_packet(&mote_id);                            /* receive response packet */
        parse_packet(&TIME, &CHANNEL, &DATA, packet);                 /* parse the packet and fetch data*/
        store_data_to_database(TIME, DATA, mote_id, CHANNEL);         /* store into database */
    }
}
```

TENET provides the API to construct a tasking packet from a task description (construct_task()), disseminate the task (send_task()), and receive the task response from the motes(receive_packet()). An application programmer only needs to write down the task description, parse the response, and process the data. In our application, the data is stored in a database, but a more realistic application might process the data in near real-time.

To describe the generality of TENET, we now show how to implement a previously proposed event-based data acquisition system, Wisden (Paek *et al.*, 2005; Xu *et al.*, 2004), in TENET. In Wisden, each mote

transmits only interesting events using a simple *onset detector* (Paek *et al.*, 2005), greatly reducing the communication requirements. [1] A TENET application *functionally equivalent* to Wisden can be implemented by simply adding a *DetectOnset* element to the task description in the above application. This application will task the motes using a task description of the form:

```
SampleMDA400(20ms, 40, A, Z_AXIS) -> DetectOnset(A, B) -> SendStr()
```

Note that all other parts of the application code can be re-used, and no modification is required in any other part of the system including the motes.

Figure 2 shows the vibration time-series on two MicaZ motes. One mote was programmed with the above task (the upper time-series) , and the other was programmed with the same task but *without* the onset detection tasklet (the lower time-series). We then put the two motes on a table, and hit it. Note that the former mote sends vibration data even before the onset; this vibration is induced by human activity (movement, typing). The latter does not (a horizontal line), and in our small experiment, results in a 90% traffic reduction.

This TENET implementation of Wisden is not only functionally equivalent to Wisden but improves over Wisden in several aspects. First, a TENET deployment is more scalable since the bandwidth capacity limit which bounded the number of nodes in Wisden is circumvented in TENET by using masters. Second, the TENET implementation of Wisden can alter application parameters (such as sampling frequency, channels, etc) at run-time by re-tasking the motes whenever the application wishes to; by contrast, whereas Wisden required re-programming of the mote nodes. Finally, in TENET, other application tasks can run concurrently while running the Wisden application. This not only provides great re-usability within the network, but it can also provide useful information about the state of network (routing topology, time-sync state, and memory usage) while an application executes. This can improve the manageability and the robustness of the sensor network.

The Networking Subsystem

TENET's networking subsystem has two simple functions: to disseminate tasks from any master to all motes, and to transport task responses from motes back to the tasking masters.

A central component of TENET's networking subsystem is one that reliably disseminates task descriptions from any masters to all motes. TENET's reliable task dissemination mechanism provides a generic API *"send_task()"* which is built upon an abstraction of reliably flooding *a sequence of packets* from *any master* to *all motes* in the network. It employs negative acknowledgment and exponential back-off timers (Levis *et al.*, 2004), to ensure reliable delivery of tasks while keeping the overhead minimal.

TENET also provides a mechanism for transporting task responses from a mote to the master that originated the task. To support various applications with different reliability requirements, TENET supports three types of delivery mechanisms: a best effort transport useful for loss-tolerant low rate applications, a transactional reliable transport for events, and a stream transport for loss-intolerant high-data rate applications (such as imaging, acoustics, and vibration data). The stream transport abstraction allows a mote to reliably send a stream of packets to a master using an end-to-end connection establishment mechanism with negative acknowledgments to repair any lost packets. Applications can select a transport mechanism by using the corresponding tasklet in their task description.

Both of above are built on top of the robust and scalable TENET routing system which can find a path between a mote and a master if there exists physical multi-hop connectivity between them. TENET uses a novel tiered routing mechanism, where a mote's response is first routed to its nearest master using a multi-sink tree-based routing, and is then routed on the master tier using an IP overlay. Our experiment used static

[1] An onset is defined to be a part of a signal waveform that exceeds the mean amplitude by more than a few standard deviations.

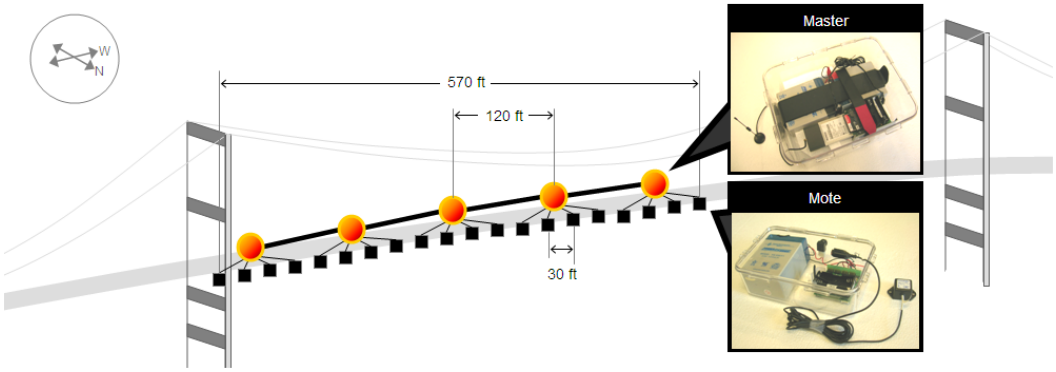

Figure 3: Deployment topology on the bridge

routing for this IP overlay for simplicity, but TENET and can easily accommodate multi-hop wireless routing protocol implementations such as Roofnet (MIT, 2005).

Finally, all pieces of our network subsystem (routing, task dissemination, and reliable transport) works over multiple hops transparently across the two tiers of the network in TENET.

Deployment: Methodology and Experiences

We recently deployed a TENET system, running a simple data acquisition application, on a 6,000 ft long suspension bridge with a main suspension span of 1,500 ft, and a height of 185 ft above water. This section describes the details of the deployment.

Hardware Setup We deployed a tiered network of Stargates and MicaZ motes. Attached to each MicaZ mote is a *vibration card* specially designed for high-quality vibration sensing. The vibration card can be programmed to sample at frequencies from 5Hz to 10KHz at 16 bits per sample and has a programmable anti-aliasing filter to accommodate different sampling rates. The 16-bit ADC of the vibration card is controlled by an on-board microprocessor, which in turn can be commanded by the attached MicaZ mote via a serial port. The stored samples can be retrieved from the on-card 64K byte SRAM by issuing commands over the serial port. Finally, we attached highly sensitive tri-axial accelerometers (dynamic range of -2g +2g, sensitivity in the μg range), capable of sensing up to three channels (3-axes) of vibration data.

To each Stargate, we attached a 5-dBi external antenna for the IEEE 802.11b wireless radio in order to extend its communication range to over 120 ft. We also used an external 8GB compact flash card to log all the incoming and outgoing packets in addition to the vibration data gathered by our application.

To conduct the experiment for over 24-hour period, we used 4 Ah batteries for the motes and 14 Ah batteries for the masters. These enabled a lifetime of up to 36 hours for the motes and over 30 hours for the masters. The master which was running the application (and hence receiving the most network traffic) was the bottleneck node for this 30 hour limit.

In planning for the deployment, we conducted a preliminary wireless communication site survey. During this survey, we noticed that the communication environment on the bridge was relatively harsh. Masters were able to communicate at distances of only 100 ft with less than 1% packet loss rate. Beyond 140 ft, they were unable to communicate. Motes were able to communicate only 80 ft reliably, and there was a sharp drop off of connectivity at 90 ft. These results were surprisingly worse than what we could achieve in a ground-level open space, and dictated our deployment topology, which we describe next.

Deployment Topology The deployment spanned 570 ft (20 motes with 30 ft between each mote) starting from the east tower towards the center of the bridge (Figure 3). The network was configured as a linear topology of 20 motes and 5 masters. Four motes were associated to each master within one communication hop, and the masters were within 2-hop distance to the center master. Each mote was placed 30 ft apart from each other resulting in a maximum distance of 45 ft from any mote to its associated master. Every pair of neighboring masters was separated by about 120 ft.

The accelerometers were attached to the pillars of the bridge with heavy-duty double-sided mounting tape, and additional duct tape wrapped around it. Each accelerometer was aligned using an off-the-shelf leveler with x-axis in north-south direction, z-axis in west-east direction, and y-axis in sky-ground direction. Each master and mote box (Figure 3) was placed on the catwalk underneath the bridge and securely taped to the bridge.

The Application On this network, we ran a TENET application which tasks all the motes to sample at 20Hz along three axes (x,y,z), retrieves the data, and stores the responses into a log file at the master. Although our hardware and the TENET system permits a sampling frequency of up to 10kHz, it is not possible to gather at such a high rate given the capacity limits on current wireless devices. To achieve a higher sampling frequency, our system must gather data no larger than the memory on the motes permit, and then retrieve them afterward.

Based on prior experiences with hardware limitations Paek *et al.* (2005), we concluded that our system would be able to support continuous-sampling and real-time transmission without any compression at 80 Hz for a single channel and one third of that for three channels. To be conservative, we selected 20 Hz tri-axis for our experiment.

Also, since a mote might encounter a temporary failure and recover after a crash, our application sends a 'reboot' task to all motes and re-tasks them to restart sampling every hour. This mechanism not only helps the motes to restart after failure but also minimizes the effect of accumulated clock skew and helps us align the samples.

To recover from any temporary failure that a master might encounter, we used a watch-dog script to reboot the stargate and re-start all the TENET software automatically. And to avoid any data losses at the master, we periodically distribute the application data file to other masters for backup.

Experiences As discussed above, we deployed 5 masters and 20 motes along the catwalk below the bridge. The complete system took 4 graduate students less than 2 hours to deploy, and the retrieval of the equipment took less than 30 minutes. Given the constraints imposed by the narrow catwalk, we believe our deployment would have been much faster at other easily accessible places.

Results

In this section we describe some preliminary results from our deployment. We examine the spectral characteristics of the time series data obtained at different locations on the bridge. We then examine the performance of the network itself, reporting on how well the network was able to deliver samples.

Data Validation

Figure 4 shows the time-series plot of the acceleration data that our system has collected from three different nodes on the bridge. All three plots are in orthogonal direction to the length of the bridge, and were taken at around 5:00pm. Also, the first two plots are from nodes at close by locations, and the last plot is from a node 300 ft away from the second one. Notice that in all three graphs, the ambient vibration levels are rather

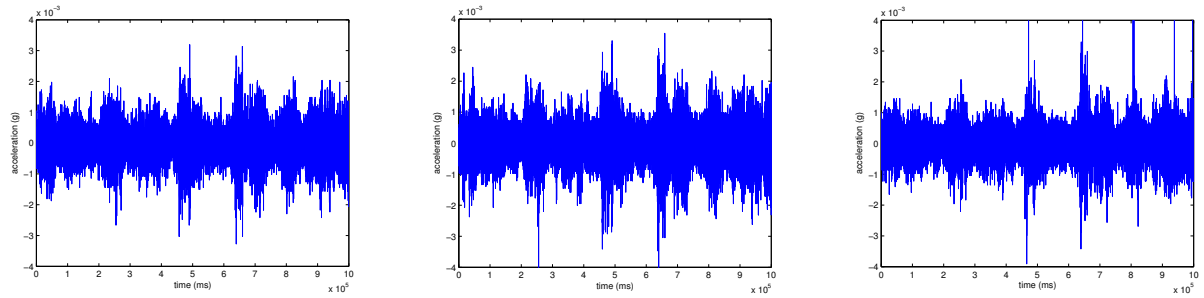

Figure 4: Time-series plot of the acceleration data from three different nodes: measured in perpendicular direction to the bridge, at around 5:00pm

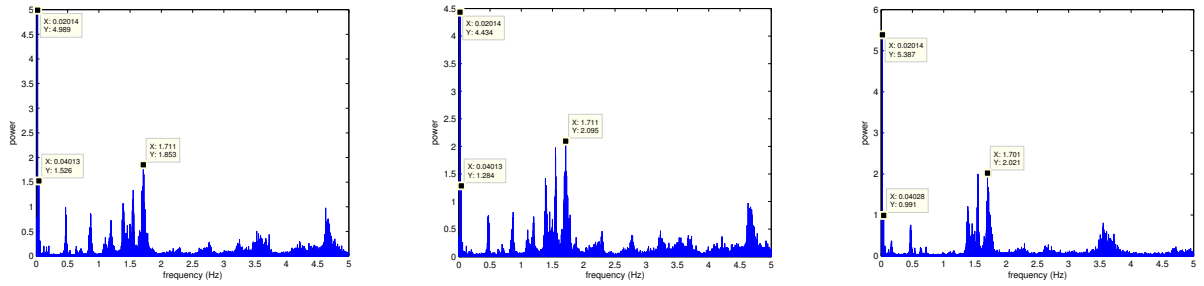

Figure 5: Frequency-domain plot of the data from three different nodes : measured in perpendicular direction to the bridge(positive acceleration facing to North), at around 5:00pm

low (in the sub-milli-g range); this is as one might expect of a large structure.

Figure 5 depicts the spectral density corresponding to the time-series snapshots in Figure 4. Despite having somewhat noisy accelerometers, readings from different sensors show consistent spectra: in all three plots, the three most dominant frequencies occur at the same values: 0.02Hz, 0.04Hz, and 1.7Hz. We have left a detailed analysis of the data set for future work.

System Evaluation

Our experiment ran for 26 hours which comprises 22 *rounds*: in each round the system collects data for 60 minutes, then replicates the collected data over the next 10 minutes. Out of 22 rounds, 19 rounds had data from all 20 nodes, and the other 3 rounds had data from 19 nodes, which results in a node-round yield of 99.32% (437 out of 440 node-rounds). During the three incomplete rounds, one node (in each of those three rounds) was not able to receive the task. This infrequent event occurs because a rebooted node does not initialize fast enough to receive the task.

We collected a total of 6,430,792 data packets during the experiment which amounts to approximately 323,745 packets from each sensor node. Each of the 19 complete data-sets have approximately 294,500 packets and 4,386,000 samples per round.

For all 437 node-rounds in which we were able to collect data from all the nodes, our system achieved 100% reliability with no lost packets. To achieve this, the end-to-end transport protocol in TENET was required to retransmit 26,965 packets, which is 0.42% of the total number of data packets received (Figure 6). Nodes on the left and the right of the figure required more retransmissions because their packets traversed longer paths (more hops) to get to the collection point which was at the center of the network.

Among those retransmissions, approximately 2.92% were due to packet loss on the IEEE 802.11b Master

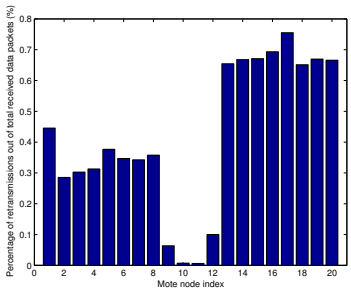

Figure 6: Percentage of retransmissions out of total number of data packets received

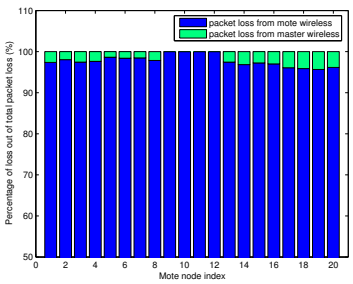

Figure 7: Percentage of retransmissions due to master links and mote links

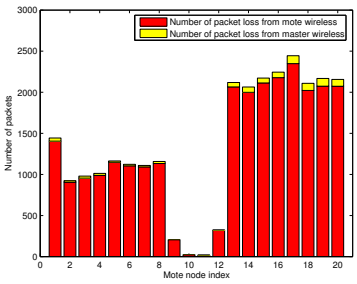

Figure 8: Number of packet retransmissions due to master links and mote links

wireless links and 97.08% were due to loss on the IEEE 802.15.4 mote wireless links (Figure 7). In Figure 8, the nodes at the center (9-12) do not show any loss in the master wireless network because the data from these motes is directly collected in the master at the center of the network while data from rest of the motes have to go over one or two master wireless links. Despite losses on the mote as well as master wireless links, our system is able to reliably collect all the transmitted samples.

Conclusion

In this paper, we have described a generic software system for implementing structural monitoring on a network of wireless sensors. Indeed, our TENET software can be used for other sensing applications as well. We have described a deployment of a network of wireless sensors on a suspension bridge, and have presented some preliminary results. We expect, in the near future, to be able to explore the structural characteristics of the bridge using a careful analysis of the dataset.

Acknowledgements

References

Doebling, S. W., Farrar, C. R., Prime, M. B., & Shevitz, D. W. 1996 (May). *Damage Identification and Health Monitoring of Structural and Mechanical Systems from Changes in their Vibration Characteristics: A Literature Review*. Tech. rept. Los Alamos National Laboratory.

Gnawali, Omprakash, Greenstein, Ben, Jang, Ki-Young, Joki, August, Paek, Jeongyeup, Vieira, Marcos, Estrin, Deborah, Govindan, Ramesh, & Kohler, Eddie. 2006 (November). The TENET Architecture for Tiered Sensor Networks. *In: Proc. ACM Conference on Embedded Networked Sensor Systems*.

Levis, Philip, Patel, Neil, Culler, David, & Shenker, Scott. 2004 (March). Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. *In: First Symposium on Network Systems Design and Implementation, NSDI'04*.

Lynch, Jerome P., & Loh, Kenneth. 2005. A Summary Review of Wireless Sensors and Sensor Networks for Structural Health Monitoring. *Shock and Vibration Digest*, **38**(2), 91–128.

MIT. 2005. *The MIT Roofnet Project*. `http://pdos.csail.mit.edu/roofnet/`.

Paek, J., Chintalapudi, K., Govindan, R., Caffrey, J., & Masri, S. 2005 (May). A Wireless Sensor Network For Structural Health Monitoring: Performance and Experience. *In: Proc. of the Second IEEE Workshop on Embedded Networked Sensors*.

Xu, N., Rangwala, S., Chintalapudi, K., Ganesan, D., Broad, A., Govindan, R., & Estrin, D. 2004 (November). A Wireless Sensor Network for Structural Monitoring. *In: Proc. ACM Conference on Embedded Networked Sensor Systems*.