Autonomous smart sensor network for full-scale structural health monitoring

Jennifer A. Rice*[a], Kirill A. Mechitov[b], B.F. Spencer, Jr.[c], and Gul A. Agha[b]
[a] Dept. of Civil and Environmental Engineering, Texas Tech University, Lubbock, TX 79409, USA;
[b] Dept. of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign
205 North Mathews Avenue, Urbana, IL USA, 61801;
[c] Dept. of Computer Science, University of Illinois at Urbana-Champaign
201 North Goodwin Avenue, Urbana, IL USA, 61801.

ABSTRACT

The demands of aging infrastructure require effective methods for structural monitoring and maintenance. Wireless smart sensor networks offer the ability to enhance structural health monitoring (SHM) practices through the utilization of onboard computation to achieve distributed data management. Such an approach is scalable to the large number of sensor nodes required for high-fidelity modal analysis and damage detection. While smart sensor technology is not new, the number of full-scale SHM applications has been limited. This slow progress is due, in part, to the complex network management issues that arise when moving from a laboratory setting to a full-scale monitoring implementation. This paper presents flexible network management software that enables continuous and autonomous operation of wireless smart sensor networks for full-scale SHM applications. The software components combine sleep/wake cycling for enhanced power management with threshold detection for triggering network wide tasks, such as synchronized sensing or decentralized modal analysis, during periods of critical structural response.

Keywords: Structural health monitoring, smart sensor networks, power management

1. INTRODUCTION

Structural health monitoring (SHM) provides the means for capturing structural response and assessing structural condition. Monitoring the safety and functionality of the world's buildings, bridges, and lifeline systems is critical to improving maintenance practices, minimizing the cost associated with repair and ultimately improving public safety. Advances in wireless technology and embedded processing have made much lower-cost wireless smart sensor networks an attractive alternative to wired data acquisition systems. The majority of the work using wireless sensors for structural monitoring has focused on using the sensors to emulate traditional wired sensor systems.[1,2,3] As these systems require that all data be sent back to a central processing center, the amount of wireless communication required in the network can become costly in terms of excessive communication times and the associated power it consumes as the network size increases. For example, a wireless sensor network implemented on the Golden Gate Bridge that generated 20MB of data (1600 seconds of data, sampling at 50 Hz on 64 sensor nodes) took over nine hours to complete the communication of the data back to a central location.[2]

Wireless smart sensors networks (WSSNs) leverage onboard computational capacity on the wireless sensors to allow data processing to occur within the network, as opposed to at a central location. By implementing data processing techniques, such as modal analysis or damage detection algorithms, in such a distributed manner, the amount of communication that occurs within the network can be reduced, while providing usable information on the structural condition.[4] WSSNs employing decentralized computing offer a scalable solution that has the potential to dramatically improve SHM efforts.

*jennifer.rice@ttu.edu; phone 1 806 239-6838; fax 1 806 742-3488

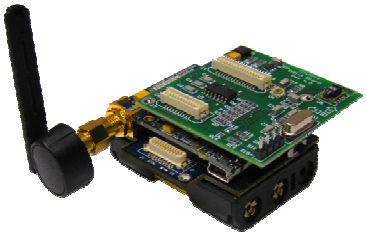
1.1. Motivation

In recent years, researchers have made tremendous strides in the development of middleware services and application services to enable SHM of civil infrastructure using WSSNs. These components include time synchronization, reliable communication, network-wide synchronized sensing, decentralized modal analysis, damage detection algorithms, etc. [5,6,7,8,9,10] This software lays the groundwork for full-scale, autonomous monitoring of civil infrastructure; however, it does not address these concerns that arise when moving from a laboratory setting to a full-scale deployment. Three critical requirements for full-scale WSSN deployments drive the smart sensor software that is presented in this paper: 1) continuous and autonomous monitoring, 2) efficient power management, and 3) data inundation mitigation. While these may appear to be conflicting goals, careful application design can meet the requirements for all three. The solution is a network that is only minimally active during non-critical structural response, but becomes fully active to measure higher response levels.

Ideally, full-scale WSSN deployments should require minimal external interaction. After some initialization involving the establishment of network operation parameters, the network should run autonomously unless instructed otherwise by the network administrator. Special care must be taken in the design of the application software to ensure a continuous and autonomous operating scenario is achieved while maintaining power efficiency. These measures can be divided into three categories: schedule-based operations, trigger-based operations, and safe-guard features. This paper presents software developments in each of these categories that, when integrated, enable full-scale, autonomous network operation.

1.2. Network components

The smart sensor platform used in this research is the Imote2 (Figure 1). The Imote2 is built around Intel's low-power X-scale processor (PXA27x). The scalability of the processor speed based on application needs allows for increased performance without a significant increase in overall power consumption. The onboard memory of the Imote2 is another feature that sets it apart from other wireless sensor platforms and allows its use for the high-frequency sampling and computationally intense data processing required for dynamic structural monitoring. It has 256 KB of integrated SRAM, 32 MB of external SDRAM, and 32 MB of flash memory.[11]



mote2 and SHM-A sensor board stack.

The Imote2 provides a flexible platform for a range of sensing applications. The sensors used with the Imote2 are interfaced to the main board via two connectors in a stackable configuration. The Imote2 does not have an onboard analog-to-digital converter (ADC) and therefore is only compatible with digital inputs, specifically I2C (a two-line serial data bus that supports multiple data channels) and SPI (a four-wire digital bus that typically only supports a single data channel). The Imote2's flexible sensor interface allows its users to tailor sensor boards to their application. While the network operation software components presented in this paper are independent of the sensor board in use, the SHM-A sensor board[12], measuring three axes of acceleration, was used during development and in the validation studies.

The proposed network configuration is homogenous from a hardware perspective, although the nodes that make up the network may be functionally differentiated, depending on the particular application that is implemented. The sensor nodes that comprise the network are called *leaf nodes*. A base station PC provides the interface to the network via the *gateway node*, which is connected to the PC via USB. Leaf nodes may be given additional responsibility beyond sensing, as described in Section 3.

2. SLEEP CYCLING

In a traditional wired sensor implementation, power management is of little concern. The sensors can remain active at all times and thus have the ability to be interrogated at any time to acquire data. Unlike such wired systems, one of the most critical features of a successful WSSN deployment is the implementation of careful power management strategies. A common approach to achieving significant energy savings in sensor network applications is to allow the sensor nodes to sleep during periods of inactivity while waking periodically to listen for instructions.[13,14] The Imote2 allows the processor to be put into a deep sleep mode, whereby only the clock component of the processor is supplied power; all other components are powered down. When the node is in the deep sleep state it cannot send data or receive via the radio or the serial ports and the LEDs do not function. Effectively, the node has no power until the sleep time expires.

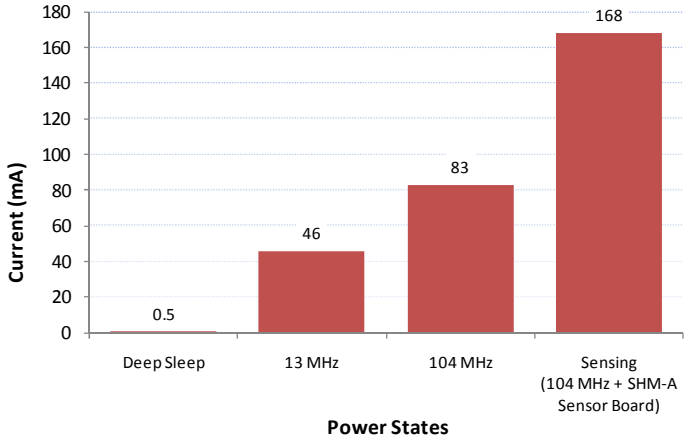


Figure 2. Imote2 and SHM-A sensor board current draw for various node operations.

The deep sleep mode lasts for a set period of time (thus the need for the clock to be powered) and results in significantly reduced power consumption. Figure 2 illustrates the power savings associated with the deep sleep mode; deep sleep consumes approximately one percent of the power consumed during listening/idle operation (13MHz).

While it may seem advantageous to keep the nodes in the deep sleep mode for extended periods of time to save power, this approach limits the ability of the gateway node to access the network at random to send inquiries or initiate network operations. To take advantage of the power savings of the deep sleep mode, while still allowing the gateway node access to the leaf nodes, a sleep/wake cycle service called *SnoozeAlarm* has been implemented. When *SnoozeAlarm* is operating on the leaf nodes (i.e. they are in the *SnoozeAlarm* mode), they sleep for a period of time, SLEEP_TIME, and then wake up for a short period of time, WAKE_TIME, during which they can listen and receive messages. This process is illustrated in Figure 3. The ratio between WAKE_TIME and the sum of WAKE_TIME and SLEEP_TIME is the *SnoozeAlarm* duty cycle. For optimal power saving, the duty cycle should be minimized while still allowing the listen time to be long enough to receive and process commands (>500 ms). The overall power saving resulting from the use of *SnoozeAlarm* depends on the particular application that is implemented and the duty cycle that is used.

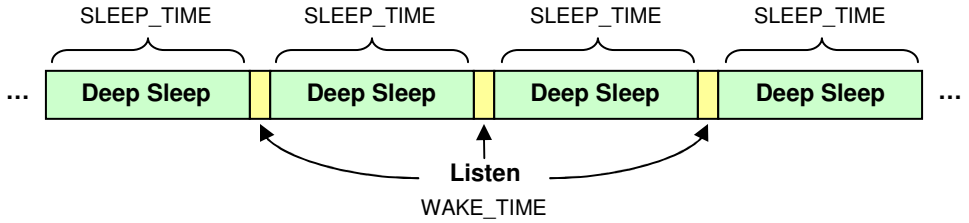


Figure 3. *SnoozeAlarm* timing.

SnoozeAlarm provides three interfaces to the application:

- `SnoozeAlarm.wakeup(targets, tcount)`: Command given to the gateway node to wake up targets (tcount is the number of targets).
- `SnoozeAlarm.awake(targets, tcount)`: Event signaled on gateway node with the node IDs and number successfully woken leaf nodes.
- `SnoozeAlarm.stayawake()`: Command given on leaf node to stay awake (i.e. stop wakeTimer).

If a message is received during the wake period, the leaf node stays awake until it is placed back in the sleep/wake cycle. *SnoozeAlarm* leverages the fact that the reliable communication protocol that ensures lossless data and guaranteed command transmission[9,15], *ReliableComm*, continuously resends packets to the destination node until it receives an acknowledgement packet. This method of communication allows the gateway node to send a wake up (or other command) to the leaf node, even if it is in the deep sleep mode. *ReliableComm* will continue to send the message until the leaf node wakes up or the message is withdrawn. Upon reception of the command, the leaf node stops its wake timer, sends an acknowledgement packet, and awaits the next command. The node resumes the *SnoozeAlarm* cycle upon being put back to sleep or being reset (either by software or a hard reset). Figure 4 illustrates how *SnoozeAlarm* operates on the leaf nodes.

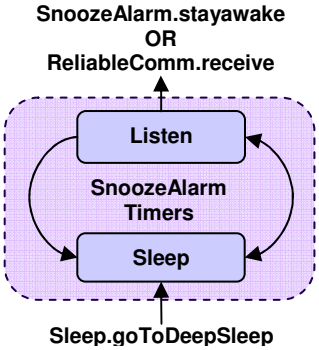


Figure 4. Leaf node *SnoozeAlarm* interfaces and operation.

The `SnoozeAlarm.wakeup` command provides an efficient method for waking a network of nodes in *SnoozeAlarm* mode from the gateway node. The *ReliableComm* protocol for broadcasting messages to a group of nodes is only successful if all of the destination nodes respond with an acknowledgement in a set period of time, thus limiting its use for waking the network. Instead the network is woken in a serial manner using successive unicast (one-to-one) commands from the gateway node to individual nodes in the network. The gateway node cycles through the list of sleeping leaf nodes, sending a wakeup command to one node in the list each time the wake-up timer fires. When a node sends back an acknowledgement, thus indicating it received the message and is remaining awake, it is removed from the list of nodes to wake up and added to the list of nodes that have been successfully woken up. This process continues until all nodes are awake or until a time-out timer expires. In both cases, a list of the nodes successfully woken up is signaled in the `SnoozeAlarm.awake` event. The wake up process is illustrated in Figure 5.

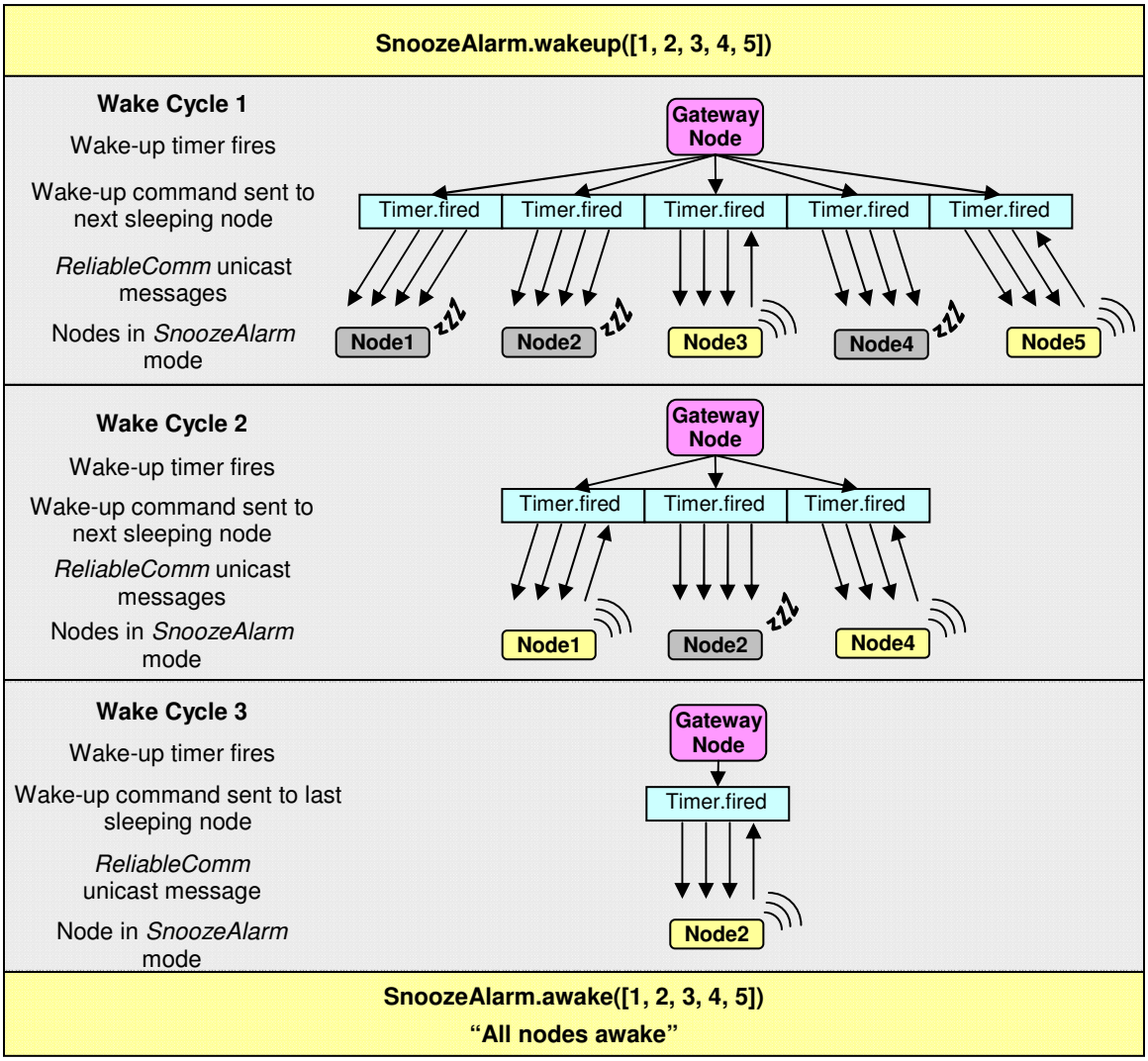


Figure 5. *SnoozeAlarm.wakeup* command

3. THRESHOLD TRIGGERING

Whether the goal is to implement centralized data collection or decentralized data processing following network sensing, it is desirable to capture the response of the structure during higher levels of excitation and motion. Higher response levels result in higher signal-to-noise ratios in the sensed data and, ultimately, more accurate results in system

identification and damage detection. One approach to ensure that important occurrences are captured by the sensor network is to designate a subset of the network to sense data on a more frequent basis than the rest of the network and provide alerts.[13,16] The *ThresholdSentry* tool allows a subset of the network to act as *sentry* nodes that are woken up periodically to sense data for a short period of time, determine if a set threshold has been exceeded in the measured data, and send a flag back to the gateway. The gateway node signals a `ThresholdSentry.exceeded` event upon reception of a flag indicating the threshold has been exceeded. This event can be used by the higher level application to make decisions on whether to wake the network and initiate network sensing or distributed modal analysis, etc. The current implementation of *ThresholdSentry* utilizes acceleration measurements; however triggers, such as strain levels or wind speed, may be incorporated into the application.

ThresholdSentry is setup on the gateway node by specifying the nodes that comprise the sentry network, the interval at which they will be asked to sense data, the duration of the data check on each sentry node, the sampling parameters for the data check, and the threshold value used for comparison in the data check. Once *ThresholdSentry* is initiated, a timer is started that fires after the check interval is reached. When this timer fires, the gateway node sends a wakeup command and sentry request to the first node in the list of sentry nodes. Upon reception of the sentry request, the sentry node senses for the prescribed period of time. When the data collection is complete, the sentry node checks the absolute maximum normalized value for each channel that collected data. The maximum peak of all the channels is then compared to the threshold value. If the threshold is exceeded, the sentry node sends a flag, with the peak measured value, back to the gateway and remains awake to wait for the next command. If the gateway receives the flag indicating the threshold has been exceeded, it signals the `ThresholdSentry.exceeded` event. If the threshold is not exceeded on the sentry node, it sends a message back that indicates the threshold was not exceeded and puts itself back into *SnoozeAlarm* mode. If the gateway node receives a message that says that the threshold was not exceeded, the gateway node restarts the check timer and moves on to the next sentry node in the queue. *ThresholdSentry* operation on the gateway and sentry nodes is illustrated in Figure 6 and the states and transitions shown in Figure 6 are described in Table 1.

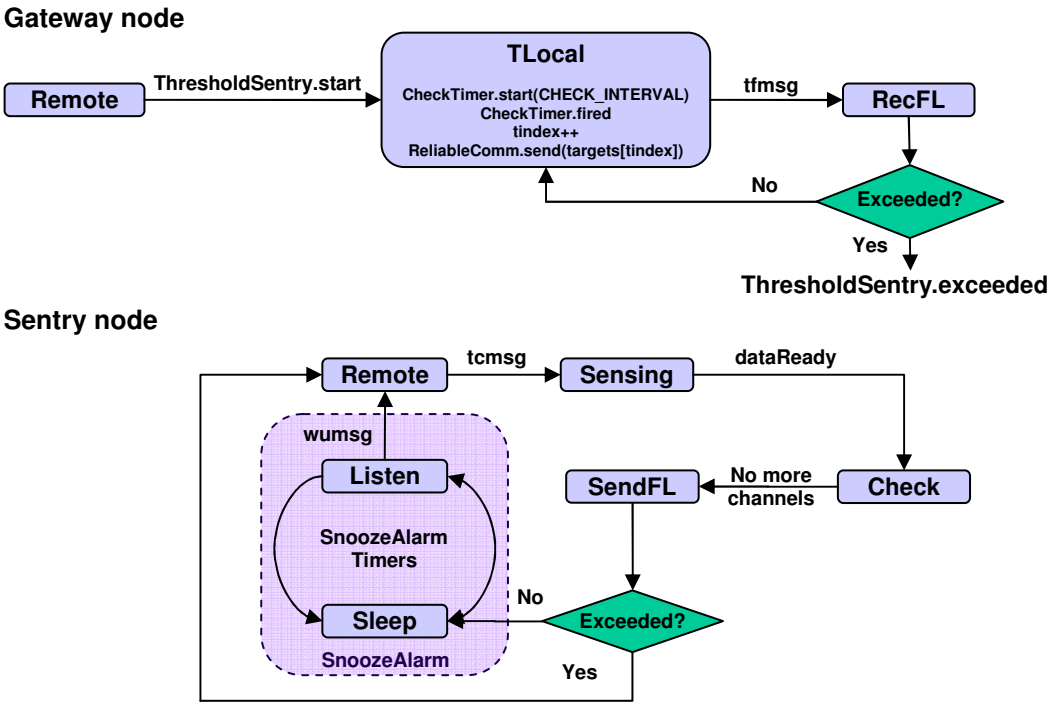


Figure 6 *ThresholdSentry* operation on gateway node (top) and leaf nodes with *SnoozeAlarm* (bottom).

Table 1. State and transitions for *ThresholdSentry*.

State	Description
Remote	Initial state
TLocal	Initial gateway node state
Sensing	Data acquisition
Check	Calculation of maximum value of normalized sensed data
SendFL	Send threshold flag
RecFL	Receive threshold flag
Transition	
ThresholdSentry.start	Application initialized by user
tcmsg	Threshold check message sent to next sentry node in queue
dataReady	Sensing on sentry node is complete
no more Channels	Peak for all channels calculated and checked against threshold
tfmsg	Threshold flag message sent from sentry node to gateway node

The selection of the threshold value and the sentry nodes within the network should be made such that the threshold is exceeded often enough for adequate structural monitoring, but not an excessive number of times at the risk of data inundation and higher power consumption levels. Because a single threshold value is used for all sentry nodes, the nodes selected as sentry nodes should measure similar levels of vibration to ensure consistency in the events that trigger the network. For example, on a long-span bridge, the nodes located near the support piers are expected to experience much lower vibration levels than those near the mid-span of the bridge. Sentry nodes in each of these locations and would exceed the threshold under very different loading circumstances. Because of additional sensing duties, sentry nodes will consume more power than non-sentry nodes. However, if more nodes make up the sentry network, the burden of increased power consumption on each sentry node will decrease because the gateway will call on each sentry node fewer times within a day. If only a few sentry nodes are selected, a larger power source may be required for those nodes. Also, the check time interval must be carefully selected so that important events are captured, while power management is still considered.

Several safeguards have been built into *ThresholdSentry* to ensure its continuous operation in spite of potentially unexpected network behavior. When a sentry request is sent to a sentry node, a timer is started on the gateway node. If the sentry node does not respond before the time expires, the gateway node prints a message that the node was not responsive and moves onto the next sentry node. This measure ensures that if a sentry node dies or becomes unavailable for some reason, the application will continue. Carefully monitoring the debug output is important to diagnose problems within the network. A sentry node that is consistently skipped indicates that it requires attention. On the sentry nodes, timers are also implemented to reset the node if it does not carry out its duty within the time allotted. This measure ensures that a node does not stay awake in an unexpected state for a long period of time draining power. In the case that the reset does take effect, a Watchdog timer will ultimately reset the node, thus ensuring that no remote node hangs indefinitely.

The current implementation of *ThresholdSentry* used in conjunction with *RemoteSensing* allows the network to record the response of longer-duration, lower-frequency events such as high wind; however, it does not support capturing short-term, transient events such as an earthquake. The time required to wake the network and perform time synchronization prior to the collection of data would cause such events to be missed.

4. AUTONOMOUS MONITORING

4.1. Network management software

Achieving an autonomous SHM implementation on a network of smart sensors requires a high-level application to coordinate each of its components in response to various events. *AutoMonitor* has been developed to provide this functionality and is described in this section. Although *AutoMonitor* can be used to initiate a variety of network wide operations, including sensing, decentralized modal analysis, damage detection, etc., the implementation described in this paper focuses on the coordination of network wide sensing and centralized data collection. The network sensing application is called *RemoteSensing*. *RemoteSensing* synchronizes the network, sends out the sensing parameters (number of channels, sampling rate, and number of data points), initiates sensing, synchronizes the acquired data, and

finally returns the data via a reliable communication protocol to the gateway node for storage on the base station PC. A detailed description of *RemoteSensing* can be found in Rice, et al.[17]

AutoMonitor is implemented on the gateway node and coordinates the following primary tasks:

- Define the *RemoteSensing* network and sensing parameters
- Define the *ThresholdSentry* sentry network
- Setup the *ThresholdSentry* parameters
- Start the *ThresholdSentry* component
- Wakeup the network and initiate *RemoteSensing* when the threshold value has been exceeded on a sentry node
- Automatically generate data files when *RemoteSensing* occurs
- Automatically generate log files of the gateway node debug output
- Enforce the maximum number of allowed network sensing events in a specified time period.

AutoMonitor is initiated via an input file from the base station PC that sets the parameters for each of the tasks it coordinates. Once started, it requires no additional input from the user. *AutoMonitor* can be stopped via a command from the PC, at which point *RemoteSensing* or other network operations can be carried out manually. *AutoMonitor* is restarted again with the input file. The input parameters are defined in Table 2. The selection of most of these parameters is highly application-dependent and will take a period of adjustment and refinement to optimize for each case. Many of these parameters have power consumption implications; their effect on power management is discussed in Rice and Spencer.[18]

Table 2. *AutoMonitor* input parameters

Input Command	**Description**	**Argument**	**Description**
RSSSetup	Remote synchronized sensing setup	nodeIDs	Array of leaf node IDs comprising the entire network
GDSetup	Sensing parameters for *RemoteSensing*	channelMask	Channels involved in network-wide sensing
		numSamples	Number of samples requested in network-wide sensing
		samplingRate	Sampling rate for network-wide sensing
SentrySetup	Setup for *ThresholdSentry*	channelMask	Channels involved in threshold check on sentry node
		samplingRate	Sampling rate for threshold check on sentry node
		checkTime	Duration of sensing for threshold check on remote node
		checkInterval	Time between sentry checks
		threshold	Value checked against in *ThresholdSentry* (in mg for accelerometer readings)
		rsmax	Maximum number of *RemoteSensing* events allowed in a given time period
		period	Time period outputting a debug log and resetting the number of sensing events
THSentryStart	*ThresholdSentry* initiation	nodeIDs	Node IDs of sentry nodes

Figure 7 provides a simple illustration of how the gateway node manages network operations in *AutoMonitor*. After the input file containing all of the parameters listed in Table 2 is read, *AutoMonitor* initiates *ThresholdSentry*. *ThresholdSentry* continues operating (moving through the list of sentry nodes at the specified interval, *checkTime*) until the threshold is exceeded on one of the sentry nodes. When the gateway node receives the flag that the threshold has been exceeded, it first checks whether the maximum number of *RemoteSensing* events, *rsmax*, in a set time period has

been reached. If *rsmax* has been reached, *ThresholdSentry* is resumed. Otherwise *AutoMonitor* sends a command to wake the network. Once all nodes are awake, or the wakeup command times out, *AutoMonitor* initiates *RemoteSensing* with the successfully woken nodes. After *RemoteSensing* completes, when all data is finished being written, *ThresholdSentry* is reinitiated. A timer runs in the background to keep track of the set time period specified in the input file. When this time period has elapsed, the count of *RemoteSensing* events, *rscount*, is set back to zero and the debug output for the last time period is saved to a file. For example, the application may be limited to two *RemoteSensing* events within 24 hours, with the debug output being written to a file once a day.

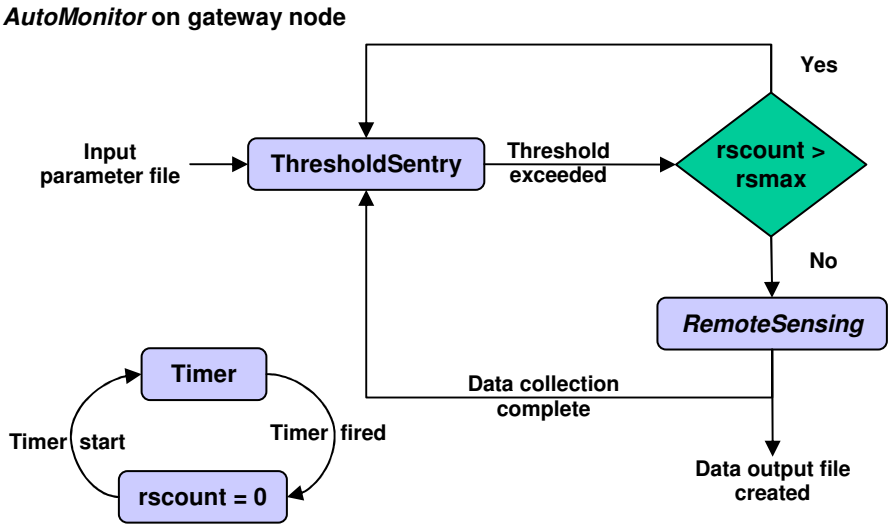


Figure 7. *AutoMonitor* operation on gateway node.

One of the safeguards built into the *AutoMonitor* applications takes advantage of the `SnoozeAlarm.awake` event. This event is signaled after the `SnoozeAlarm.wakeup` command is executed. The arguments of the `awake` event are the nodes that were successfully woken up. *AutoMonitor* initiates network sensing with the nodes that were responsive to the wakeup command, ensuring greater probability of successful network sensing. In this way, the wakeup command acts to establish the nodes that should be included in network-wide operations.

4.2. Full-scale implementation

The *AutoMonitor* network management application with *SnoozeAlarm* and *ThresholdSentry* features is employed in the ongoing monitoring of the Jindo Bridge, a long-span, cable-stayed bridge in South Korea. The Jindo Bridge deployment consists of 70 Imote2s and SHM-A sensor boards divided into two sub-networks, one with 33 leaf nodes and one with 37 leaf nodes. The two initial deployment phases were aimed at optimizing the network parameters based on the structural response and the network performance. The parameters used in each phase are summarized in Table 3. Jang, et al. discuss the details of the parameter optimization and report stable performance of the *AutoMonitor* application following some parameter adjustments.[19]

Table 3. Jindo Bridge network parameters during the first two phases of deployment.

Category	Parameter	1st deployment	2nd deployment
Snooze Alarm	*SnoozeAlarm* wake/listen time	750 ms	750 ms
	SnoozeAlarm sleep time	15 sec	15 sec
	SnoozeAlarm duty cycle	4.76%	4.76%
ThresholdSentry	Sentry network size	2	3
	ThresholdSentry check interval	20 min	10 min
	ThresholdSentry sensing time	10 sec	10 sec
	Acceleration Threshold value	10 mg	10 mg
AutoMonitor	*RemoteSensing* events allowed per day	1	4

5. CONCLUSION AND FUTURE WORK

This research addresses the smart sensor software requirements for achieving autonomous, full-scale, and potentially extended WSSN deployments. Such deployments demand effective handling of resources, decision making on the pertinent data to extract from the network, and autonomous management software to control the network for extended periods of time with minimal external interaction. Three applications have been developed and integrated to achieve these goals. A sleep/wake cycle, *SnoozeAlarm*, allows the network to be in a very low power state the majority of the time it is not engaged in SHM applications, while waking periodically to listen for commands from the base station. The power savings associated with this sleep/wake cycle are critical for achieving WSSN deployments beyond just a few days when battery power is utilized. A second service, *ThresholdSentry*, is deployed on a subset of the nodes in the network to alert the gateway node when measured data is at a level that warrants network-wide action, such as sensing alone or sensing in combination with distributed data processing strategies. This service limits data acquisition and communication to events of interest, thus minimizing unnecessary data communication and processing while conserving network resources. The *AutoMonito*r network management service provides over-arching and continuous control of network functionality, including the establishment of network parameters, the operation of *ThresholdSentry*, the wake up and initialization of network wide synchronized sensing in response to critical loading, and data file generation. The results from Jindo Bridge represent the first autonomous, large-scale deployment of a network of smart sensors for structural monitoring. The framework provides the basis for future implementation of distributed data processing, which will further propel smart sensor technology for SHM applications. All software described in this paper is open source and available at http://shm.cs.uiuc.edu/software.html.

The current implementation of *ThresholdSentry* does not support real-time triggering of network-wide sensing in reaction to a short-term, high structural response levels. The primary limitations are the time it takes to wake the network (when it is in the *SnoozeAlarm* mode) and the time associated with network time synchronization. In a large network, such as the one deployed on Jindo Bridge, it can take over one minute from a sentry node sending an alert to the actual start of data acquisition. In the event of an earthquake the entire occurrence could be missed while the network is initiated. Solutions to achieve the ability to capture such an event will likely require an integrated hardware/software approach and are the subject of ongoing research.

6. ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of this research by the National Science Foundation, under grant CMS 0600433 (Dr. S. C. Liu, program manager).

REFERENCES

[1] Arms, S. W., Galbreath, J. H., Newhard, A. T., and Townsend, C. P., "Remotely reprogrammable sensors for structural health monitoring," *Smart Materials Technology: NDE/NDT for Highways and Bridges* (2004).

[2] Pakzad, S.N., Fenves, G.L., Kim, S., Culler, D.E., "Design and Implementation of Scalable Wireless Sensor Network for Structural Monitoring," *ASCE Journal of Infrastructure Engineering*, 14(1): 89-101 (2008).

[3] Whelan, M.J and Janoyan, K.D., "Design of a Robust, High-rate Wireless Sensor Network for Static and Dynamic Structural Monitoring," *J. Intelligent Material Systems and Structures* 20 (7): 849-863 (2009).

[4] Nagayama, T., Sim, S-H., Miyamori, Y., and Spencer Jr., B.F., "Issues in structural health monitoring employing smart sensors," *J. Smart Structures and Systems*, 3(3): 299-320 (2009).

[5] Nagayama, T., Spencer Jr., B.F. and Rice, J.A., "Autonomous Decentralized Structural Health Monitoring Using Smart Sensors," *Structural Control and Health Monitoring* 16(7-8): 842-859 (2009).

[6] Rice, J.A., Mechitov, K.A., Sim, S. H., Spencer Jr., B.F. and Agha, G., "Enabling Framework for Structural Health Monitoring Using Smart Sensors," *Structural Control and Health Monitoring* (accepted).

[7] Sim, S.H. and Spencer, Jr., B.F., "Decentralized Strategies for Monitoring Structures using Wireless Smart Sensor Networks," *NSEL Report Series, No. 19*, University of Illinois at Urbana-Champaign, http://hdl.handle.net/2142/14280 (2009).

[8] Mechitov, K., Kim, W., Agha, G., and Nagayama, T., "High-Frequency Distributed Sensing for Structure Monitoring", *Proc. First Intl. Workshop on Networked Sensing Systems*, Tokyo, Japan, 101–105 (2004).

[9] Nagayama, T., Spencer Jr., B.F., Mechitov, K., Agha, G. "Middleware services for structural health monitoring using smart sensors, *Smart Structures and Systems*," 5(2) (2009).

[10] Castaneda, N., Sun, F., Dyke, S., Lu, C., Hope, A., Nagayama, T. "Implementation of a Correlation-based Decentralized Damage Detection Method Using Wireless Sensors," *Proc. 2008 ASEM Conference*, Jeju, Korea, (2008).

[11] Crossbow Technology, Inc., "Imote2 Hardware Reference Manual," San Jose, CA (2007).

[12] Rice, J.A, Mechitov, K., Sim, S.H., Nagayama, T., Jang, S., Kim, R., Spencer Jr., B.F., Agha, G. and Fujino, Y., "Flexible Smart Sensor Framework for Autonomous Structural Health Monitoring," *J. Smart Structures and Systems* (accepted).

[13] Wang, L. and Xiao, Y., "A Survey of Energy-Efficient Scheduling Mechanisms in Sensor Networks," *Mobile Networks and Applications*, 11(5) (2006).

[14] Ye, W., Heidemann, J., and Estrin, D., "An energy-efficient MAC protocol for wireless sensor networks," *Proc. IEEE INFOCOM*, 1567-1576 (2002).

[15] Nagayama, T. and Spencer Jr., B.F., "Structural health monitoring using smart sensors," *NSEL Report Series Report No. 1*, University of Illinois at Urbana-Champaign, http://hdl.handle.net/2142/3521 (2009).

[16] Hui, J., Ren, Z., and Krogh, B.H., "Sentry-based power management in wireless sensor networks," *Proc. IPSN*, 458–472 (2003).

[17] Rice, J.A., Mechitov, K.A., Sim, S. H., Spencer Jr., B.F. and Agha, G., "Enabling Framework for Structural Health Monitoring Using Smart Sensors," *Structural Control and Health Monitoring* (accepted).

[18] Rice, J.A. and Spencer, B.F., "Flexible Smart Sensor Framework for Autonomous Full-scale Structural Health Monitoring," *NSEL Report Series, No. 18*, University of Illinois at Urbana-Champaign, http://hdl.handle.net/2142/13635 (2009).

[19] Jang, S., Jo, H., Cho, S., Mechitov, K., Rice, J.A., Sim, S.H., Jung, H.J., Yun, C.B., Spencer, Jr., B.F. and Agha, G. (2010). "Structural Health Monitoring of a Cable-stayed Bridge Using Smart Sensor Technology: Deployment and Evaluation," *J. Smart Structures and Systems* (accepted).