Smart Structures and Systems, Vol. 5, No. 2 (2009) 119-137
DOI: http://dx.doi.org/10.12989/sss.2009.5.2.119

Middleware services for structural health monitoring using smart sensors

T. Nagayama*

Department of Civil Engineering, University of Tokyo 7-3-1, Hongo, Bunkyo-ku, Tokyo 113-8656, Japan

B. F. Spencer, Jr.‡

Department of Civil Engineering, University of Illinois at Urbana-Champaign 205 N. Mathews Ave. Urbana, Illinois 61801, USA

K. A. Mechitov and G. A. Agha

Department of Computer Science, University of Illinois at Urbana-Champaign 201 N. Goodwin Ave. Urbana, Illinois 61801, USA

(Received February 1, 2008, Accepted November 12, 2008)

Abstract. Smart sensors densely distributed over structures can use their computational and wireless communication capabilities to provide rich information for structural health monitoring (SHM). Though smart sensor technology has seen substantial advances during recent years, implementation of smart sensors on full-scale structures has been limited. Hardware resources available on smart sensors restrict data acquisition capabilities; intrinsic to these wireless systems are packet loss, data synchronization errors, and relatively slow communication speeds. This paper addresses these issues under the hardware limitation by developing corresponding middleware services. The reliable communication service requires only a few acknowledgement packets to compensate for packet loss. The synchronized sensing service employs a resampling approach leaving the need for strict control of sensing timing. The data aggregation service makes use of application specific knowledge and distributed computing to suppress data transfer requirements. These middleware services are implemented on the Imote2 smart sensor platform, and their efficacy demonstrated experimentally.

Keywords: structural health monitoring, smart sensor, middleware services, synchronized sensing, reliable communication.

1. Introduction

Dense arrays of smart sensors have the potential to improve Structural Health Monitoring (SHM) dramatically using on-board computational and wireless communication capabilities. Information from dense measurements is expected to result in deeper insight into the physical state of a structural system.

Though smart sensors have seen several demonstrative applications (Lynch and Loh 2006, Kim, *et al.*

*E-mail: nagayama@bridge.t.u-tokyo.ac.jp
‡E-mail: bfs@uiuc.edu

2007), detailed validation of the data from the smart sensor networks and subsequent use for damage detection has been limited. Measured signals are oftentimes lost during wireless communication or are not synchronized with each other, hindering subsequent analyses. Slow communication speeds have prohibited data collection from spatially dense and temporally frequent measurements. (Nagayama, *et al*. 2007a). Currently available standardized wireless solution such as WLAN and Bluetooth systems do not meet SHM requirements for smart sensor networks, such as low power consumption, scalability to large numbers of sensor nodes, and precise synchronization. Sensing capabilities need to be provided that meet the demands of SHM applications, while respecting smart sensor hardware limitation.

This paper provides the necessary capabilities for implementing SHM on smart sensor networks by developing corresponding middleware services. Among the middleware services are (a) reliable communication service involving only a few acknowledgement packets, (b) synchronized sensing without the need for strict sensing timing control, and (c) efficient data aggregation with reduced data transmission demands. These middleware services are implemented on a smart sensor platform, the Imote2, which is designed for data-intensive applications such as SHM. Note that the middleware services are generally applicable to other smart sensor platforms possessing similar hardware characteristics as the Imote2. The implemented middleware services are evaluated and validated experimentally.

2. Background and motivation

This section provides necessary background and motivation for the middleware services discussed in subsequent sections of the paper.

2.1. Reliable data transfer

Wireless communication is intrinsically lossy. Communication packets are dropped due to various reasons, including: (a) small signal-to-noise ratio of received signals and (b) packet collision due to simultaneously transmitting nodes. In SHM applications, packet losses result in loss of commands and measurement data. Loss of command packets may result in losing control of sensor network operation especially for SHM systems involving complex internode data processing. The effect of data loss was theoretically investigated for spectral density estimation applications; data loss was found to degrade measurement signals in a similar way to observation noise (Nagayama, *et al*. 2007a). Because the loss of a large number of packets is not necessarily infrequent (Nagayama & Spencer 2007), packet loss compensation is required.

Redundant transmission and acknowledgment-based approaches are candidates to reduce the packet loss rate during wireless communication. For completeness, these two approaches are first explained briefly.

Redundant transmission without acknowledgment can statistically improve the reliability of communication. If the packet loss rate is expected to be approximately constant over time, this approach can virtually eliminate data loss; the degree of redundancy can be adjusted based on measured packet loss rates. In smart sensor networks, however, burst packet loss undermines the effectiveness of this approach. For example, interference from other nearby RF transmission devices operating in the same frequency range can cause burst loss. The loss of a large amount of data due to burst loss cannot be completely compensated by such statistical approaches.

Acknowledgment-based approaches, on the other hand, can guarantee reliable communication between nodes, by continuing packet retransmission until acknowledgment is received. However, a poorly-designed communication protocol involving many acknowledgment messages is notably inefficient; interwoven transmission and reception of acknowledgement-based approaches may result in slow data transfer. Instead of acknowledging each packet, the reliable communication protocol developed by Mechitov, *et al*. (2004) sends a set of packets and then waits for acknowledgment. If the sender does not receive acknowledgment, the same set of packets is sent again. Upon reception of acknowledgment, the sender moves on to the next set of packets. The size of a set of packets needs to be optimized. Therefore, the number of acknowledgement packets cannot be drastically reduced, limiting the efficiency. A reliable communication protocol involving a smaller number of acknowledgement packets is needed to improve the communication efficiency.

2.2. Synchronized sensing

The lack of a global clock in smart sensor network is problematic for SHM applications. If modal analysis is conducted with random and inaccurate time synchronization, the identified mode shapes will be inaccurate, possibly falsely indicating structural damage. Large synchronization errors make accurate estimation of cross spectral densities and transfer functions almost impossible.

Time synchronization for smart sensor networks has been widely investigated. Reference Broadcast Synchronization (RBS; Elson, *et al*. 2002), Flooding Time Synchronization Protocol (FTSP; Maroti, *et al*. 2004), and Timing-sync Protocol for Sensor Networks (TPSN; Ganeriwal, *et al*. 2003) are among the well-known synchronization methods. Mica2 motes employing TPSN are reported to synchronize with each other to an accuracy of 50 ms. Mechitov, *et al*. (2004) implemented FTSP on Mica2 motes as a part of wireless data acquisition system for SHM. This system can maintain better than 1ms synchronization accuracy for several minutes. When clock drift is significant, synchronization is applied periodically before the error becomes too large. Thus, fine synchronization among sensor nodes has been shown achievable.

However, measured signals may not be synchronized with each other, even when clocks are precisely synchronized. Sampling timing is not necessarily controlled precisely based on these clocks. Precise timing control on a sensor node was pursued and demonstrated by Kim, *et al*. (2007). Nevertheless, such approaches can be computationally expensive and sometimes impractical. For example, processes other than sensing need to be turned off to achieve precise sensing timing control while clock drift compensation in this approach requires on-the-fly calculation of drift. Synchronized sensing without the need of strict timing control is preferable.

2.3. Data aggregation

The amount of data typically collected by SHM applications employing wired sensors normally exceeds practical communication capabilities of smart sensor networks. Central data collection of 80 seconds of data is reported to take several hours (Kim, *et al*. 2007). One approach to overcome this problem has been for each node to employ an independent data processing strategy. However, this approach (i.e., without internode communication) cannot fully exploit information available in the sensor network (e.g., spatial information is neglected). Distribution of data processing and coordination among smart sensors is considered to play a central role in addressing this data aggregation issue. Application specific knowledge can be incorporated into data aggregation strategies to efficiently collect information from smart sensor network.

2.4. Intel Imote2 smart sensor platform

The Imote2 is a new smart sensor platform developed for data-intensive applications (Crossbow, Inc., 2009). The main board of the Imote2 incorporates a low-power Xscale processor, the PXA271, and an 802.15.4 radio, ChipCon 2420. The processor's frequency is scalable, thereby improving its power efficiency. One of the important characteristics of the Imote2, which separates it from other commercially available wireless sensing nodes, is the available memory size. The Imote2 has 256 KB of integrated SRAM, 32 MB of external SDRAM, and 32 MB of Strataflash memory, which is particularly important for the large amount of data required for dynamic monitoring of structures and associated data processing. These hardware specifications far exceed those of other academic and commercial wireless sensing units including MicaZ (Lynch and Loh 2006). The ITS400 sensor board (Crossbow, Inc., 2009) can measure 3-axes of acceleration, light, temperature, and relative humidity. All of the sensors on this board are digital, A four-channel 12-bit ADC is also implemented on the board so that the Imote2 can utilize external analog sensor outputs is required. The 3-axis digital accelerometer (LIS3L02DQ; STMicroelectronics, 2008) has a ±2 g measurement range and a resolution of 12-bits or 0.97 mg.

The Imote2 running TinyOS with the ITS400 sensor board are employed as the smart sensor platform in this research. The RAM size allows for buffering a large amount of data in reliable communication. Also the RAM and the fast CPU enable data processing approaches for realizing synchronized sensing, without the need for a strict hardware timing control, as well as efficient data aggregation utilizing in-network data processing. The development and experimental verification of the three middleware services are explained in the following sections.

3. Reliable communication

Reliable communication protocols for long data records and for single packet commands are developed in this section. In SHM applications, most communication takes place as unicast. Nonetheless, multicast of command packets is oftentimes needed. Some SHM applications can efficiently achieve data aggregation, taking advantage of multicast of long data records as explained subsequently. Additionally, debugging of in-network data processing becomes much easier if in-network communication packets are also multicast to the base station. Both unicast and multicast protocols are, therefore, supported in the proposed protocols.

3.1. Communication protocol for long data records

To reduce the number of acknowledgement packets involved in long data transfer, reliable communication protocols which send the entire data before waiting for acknowledgement packets are proposed. While Kim, *et al*. (2007) also employs a similar approach, their system is a receiver initiated protocol assuming central data acquisition. The protocols proposed herein assume in-network data processing, where data transfer is not necessarily one-way or unicast. Also the large RAM of the Imote2 is utilized to buffer data in a power-efficient manner.

3.1.1. Unicast

In the unicast protocol, the sender reliably delivers long data records to a single receiver node. Figure 1 shows a flowchart of this protocol. The sender first performs a handshake and transfers the necessary parameters. Then all the data is sent without requesting acknowledgements. After this transfer, the receiver checks for missing packets and requests retransmission. Following retransmission, the handshake is terminated. The details of the protocol are explained in the following paragraphs.

3.1.1.1. Handshaking

The sender first sends a packet and informs the receiver of the necessary parameters (*e.g.*, data length, data type, message ID, destination node ID, source node ID, etc.). If the data is in integer format, a scaling factor and offset of the data are also conveyed to the receiver so that the receiver can reconstruct the original data. All these parameters are packed in the 28 byte payload of the packet. On reception of this packet, the receiver node replies with acknowledgement unless this node has already been communicating with another node. If this node is already communicating, the packet reception is ignored. The handshake request packet is sent periodically until an acknowledgement from the receiver is returned. Once the acknowledgement packet is received and two nodes engage in this round of reliable communication, packets with different message IDs, destination node IDs, or source node IDs are disregarded. The message ID of the sender is incremented when the sender engages in a new round of reliable communication.

For Imote2 nodes to appropriately interpret the received packet, 1 byte of the payload of every packet is designated for packet interpretation instruction. For example, a packet from the sender with the instruction ‘1’ is interpreted as the first packet with parameters such as data length and data type. On reception of a packet, tasks predetermined for the instruction are executed.

Also, the current state of the sender and receiver are internally maintained using a 1-byte variable on each node. For example, the sender’s current state is ‘1’ after sending the packet with the instruction ‘1’. This state changes to ‘2’ when the sender finds that an acknowledgment for this packet is received. The tasks to be executed in events such as packet reception and timer firing are determined based on the current state variable and the instruction byte of the most recently received packet.

3.1.1.2. Data transfer

When the sender notices that the acknowledgment has arrived, the entire data record is sent to the

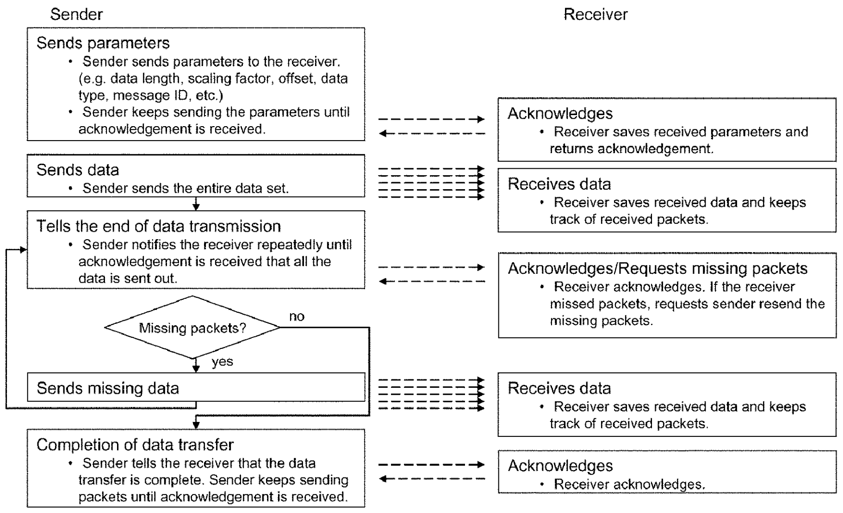
Sender
Receiver
Sends parameters
• Sender sends parameters to the receiver. (e.g. data length, scaling factor, offset, data type, message ID, etc.)
• Sender keeps sending the parameters until acknowledgement is received.
Acknowledges
• Receiver saves received parameters and returns acknowledgement.
Sends data
• Sender sends the entire data set.
Receives data
• Receiver saves received data and keeps track of received packets.
Tells the end of data transmission
• Sender notifies the receiver repeatedly until acknowledgement is received that all the data is sent out.
Acknowledges/Requests missing packets
• Receiver acknowledges. If the receiver missed packets, requests sender resend the missing packets.
Missing packets?
no
yes
Sends missing data
Receives data
• Receiver saves received data and keeps track of received packets.
Completion of data transfer
• Sender tells the receiver that the data transfer is complete. Sender keeps sending packets until acknowledgement is received.
Acknowledges
• Receiver acknowledges.

Fig. 1 Block diagram of the reliable unicast protocol for long data records

receiver. One packet contains either three double precision data points, six 32-bit integers, or twelve 16-bit integers. Each packet also contains the 2-byte node ID of the sender and the 2-byte packet ID, which the receiver uses to sort received packets and reconstruct the original data in the buffer. To keep track of received packets, the receiver maintains a bitmap of received packets. One bit of the bitmap is assigned to each data packet. Upon reception of each packet, the corresponding bit is changed to one. By examining this bitmap, the receiver determines which packets are missing.

3.1.1.3. Detection of missing packets and retransmission

When the sender finishes transmitting all the data, a packet with an instruction byte of '2' conveying the end of the data transmission is sent to the receiver. The sender periodically send this packet until an acknowledgement is received. Knowing that all of the data was transmitted from the sender, the receiver checks the bitmap. If the receiver notices that some packets are missing, the receiver responds by sending a packet with the missing packets' IDs. The first eight missing packets' IDs are packed in the payload of an acknowledgment packet. The sender then retransmits these packets. At the end of transmission, the packet with the instruction-byte of '2' is sent again. If there are still missing packets, the same procedure is repeated.

3.1.1.4. Completion of communication

When all of the packets are transferred, the sender tells the receiver that the data transfer is complete by sending a packet with the instruction-byte of '4'. This message is also retransmitted until an acknowledgment packet is returned from the receiver. When the receiver first acknowledges the packet, the receiver judges that this set of receiving activities on this node has been completed. When the sender node receives this acknowledgment, the sender completes this set of data transfer. The handshake is called off and the two nodes are ready to start another round of data transfer.

3.1.2. Multicast

In the multicast communication protocol, the sender reliably transmits long data records to multiple receivers utilizing acknowledgment packets. At the beginning of communication, the sender searches for receiver nodes. Then handshaking, data transfer, and retransmission follow before calling off the handshake, as in the unicast communication protocol. The differences from the unicast protocol are explained herein.

3.1.2.1. Search for receivers and handshaking

The primary difference between the multicast and unicast protocols is the first step of searching for the receivers. Before the sender transmits parameters such as data length and data type, the receiver node IDs, source node ID, and message ID are broadcast. The instruction byte of this packet is 'f' in hexadecimal. A maximum of eight receiver node IDs are sent in a single packet. When data is sent to more than eight nodes, multiple packets containing the receiver node IDs are broadcast. As is the case for unicast communication, the sender periodically resends these packets until the receivers reply.

Upon reception of this initial packet, receivers compare their own node IDs and those contained in the packets. If the node is one of destination nodes, this node returns an acknowledgment to the sender and commits itself to this round of communication. Once the sender receives acknowledgements from all the receivers, the sender moves to necessary parameter transfer and then to data transfer as in the unicast protocol.

3.1.2.2. Acknowledgement

Note that if multiple receivers reply to the sender at the same time, packet collision may take place; therefore, timing for replies to the sender needs to be scheduled appropriately. In this protocol, the receivers reply in the order of node IDs in the packets with the instruction-byte of 'f'. For example, the node corresponding to the eighth node ID in the 'f' packet waits $7\times\beta$ seconds before the acknowledgment is sent back to the sender; β is unit waiting time. The order of reply and β are stored on each receiver; all of the following acknowledgment messages in this multicast protocol use this scheduling. Optimization of the waiting interval, β, may result in faster communication.

Although the broadcast transmission reaches all of the nodes in the neighborhood, only the engaged nodes process the received packet. The source ID, the destination ID, and the message ID are used on receivers to distinguish packets to be processed from other packets.

3.1.2.3. Acknowledgement reception check

The sender needs to check the acknowledgment messages from all of the receivers. The sender keeps track of acknowledgments from the receivers using a bitmap. One bit of the bitmap corresponds to one receiver. Before the sender transmits a packet needing acknowledgement from the receivers, this bitmap is initialized to zeroes. Upon reception of an acknowledgment from a receiver, the corresponding bit is changed to one. The sender periodically sends an acknowledgement request until all the bits become one.

3.1.2.4. Detection of missing packets and retransmission

Retransmission is performed not for each receiver but for all of the receivers. After sending the end-of-transmission notice packet to the receivers, the sender waits for replies from the respective receivers. Within the assigned time slot, each receiver sends back an acknowledgment packet containing missing packet IDs. After the time assigned for all of the nodes to reply has passed, the sender broadcasts the requested packets. At the end of broadcast of these requested packets, a packet with the instruction-byte of '2' is sent, asking for packets still missing.

3.2. Communication protocol for commands

Communication protocols suitable to transfer single packet commands reliably are also needed. Appropriate unicast and multicast reliable communication protocols are developed.

In the unicast protocol, the sender transmits a packet, including the destination ID, source ID, message ID, command, and other parameters periodically until an acknowledgement packet from the destination node is received. The payload of one packet is 19 bytes in addition to communication parameters such as source and message IDs.

The multicast protocol is implemented in a similar manner to the unicast protocol. The primary difference is the first step of searching for the receivers. The sender first transmits a packet including the receivers' node IDs. Once all of the receivers return acknowledgment packets to the sender, the sender transmits a packet containing a command and associated parameters periodically until all the receivers return acknowledgement packets. Acknowledgement return scheduling and reception checks are performed in a similar manner to the multicast protocol for long data records.

3.3. The common issue of communication completion judgement

Receivers cannot reliably determine the end of communication. This difficulty is also known as the

Byzantine Generals Problem (Lamport, *et al.* 1982). The sender can clearly determine the end of communication when an acknowledgment packet to the communication completion notice packet, which is the packet with the instruction byte of '4' in the long data record protocol case, is received. However, the receiver cannot. The receiver does not know whether the acknowledgment reaches the sender or not. After the transmission of this acknowledgment packet, there are two possible situations: (a) the acknowledgment packet reached the sender and this data transfer was successfully completed; and (b) the acknowledgment packet did not reach the sender and the sender is periodically transmitting packets with the instruction-byte of '4', none of which had arrived at the receiver node. There is no way for the receiver node to judge between these two possibilities.

This inability to judge the end of communication may cause serious problems. If only the receiver calls off the handshake, then the sender will keep sending the last packet to the receiver; the receiver never replies because the receiver is already disengaged from this round of data transfer.

This problem is alleviated by storing message and source IDs for the last several rounds of reliable communication and separating the final acknowledgment process from the other communication processes. When the communication completion notice packet is received for the first time, this node sends an acknowledgment back and is disengaged from this communication activity. The message and source IDs are stored on this node. In case of multicast communication, the waiting time before returning an acknowledgement packet is also stored. If this receiver node later receives the communication completion notice packet again, the message and source IDs are examined. Once the receiver finds in the memory the pair of the message and source IDs that are the same as those of the received packet, an acknowledgment is sent back to the sender. Even when a node is engaged in another round of communication, this acknowledgement process separated from the others can be executed independently.

4. Synchronized sensing

Synchronized sensing without the need of strict timing control is proposed utilizing a resampling approach. The accuracy of the Imote2's time synchronization is first estimated and then issues critical to synchronized sensing are investigated. Finally, synchronized sensing is realized by extending a resampling method and combining the method with accurate time synchronization and time stamping of the Imote2 (Nagayama & Spencer 2007, Nagayama, *et al.* 2006a, 2007a, & 2007b, Spencer and Nagayama, 2006).

4.1. Estimation of time synchronization error

The accuracy of the Flooding Time Synchronization Protocol (FTSP; Maroti, *et al.* 2004) implementation on the Imote2 is evaluated. This implementation is first reviewed briefly, and then the accuracy is examined in terms of synchronization errors and clock drift.

FTSP utilizes time stamps on the sender and receivers. A beacon node broadcasts a packet to the other nodes. At the end of the packet, a time stamp, t_{send}, is appended just before transmission. Upon reception of the packet, the receivers stamp the time, $t_{receive}$, using their own local clocks. The delivery time, $t_{delivery}$, between these two events includes interrupt handling time, encoding time, and decoding time. $t_{delivery}$ is usually not small enough to be ignored; the variation of $t_{delivery}$ in time is usually small. In this study, $t_{delivery}$ is first assumed to be zero and then adjusted so that Imote2s placed on the same shake table give synchronized acceleration data. Using this value, the offset is determined as $t_{receive}-t_{send}-$

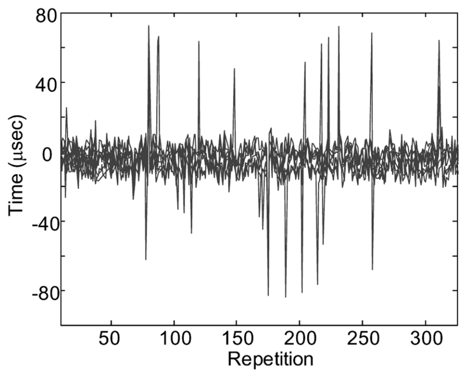

Fig. 2 Time synchronization error estimation

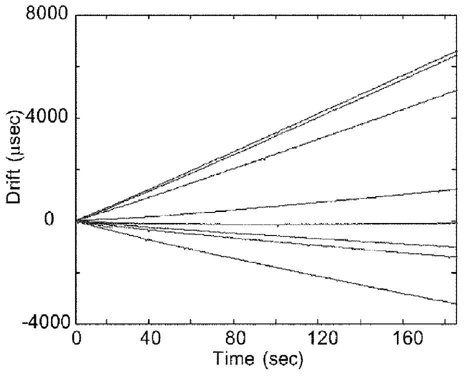

Fig. 3 Drift estimation

$t_{delivery}$. This offset is subtracted from the local time when global time stamps are requested.

To evaluate time synchronization error, a group of nine Imote2s are programmed as follows. The beacon node transmits a beacon packet to perform time synchronization every 4 seconds. Two seconds after the beacon signal, the beacon node sends another packet, requesting replies. The receivers obtain time stamps on reception of this packet and convert them to a global time stamp using the offset estimated 2 seconds before. These time stamps are subject to two error sources: (a) time synchronization error, and (b) delay in time stamping upon reception of the second packet. The receivers take turns to report back these time stamps. This procedure is repeated more than 300 times.

Time stamps from the eight nodes are compared in Figs. 2 and 3. Fig. 2 shows the difference in the reported global time stamps using one of the eight nodes as a reference. The time synchronization error is predominantly less than 10 μs. Scattered peaks may indicate large synchronization error. Note that the time synchronization error is one of the two above-mentioned factors explaining the error in the time stamps. An upper-bound estimate of time synchronization error is estimated as 80 μs.

The same approach is utilized to estimate clock drift. Upon reception of the second packet, which requests replies, the receivers return their offsets (instead of global time stamps) to the sender to estimate the global time. If clocks on the nodes were ticking at exactly the same rate, the offsets should be constant. Fig. 3 shows the offset time histories of eight receiver nodes. This figure shows that the clock drift is not negligible, and the drift rate is quite constant in time. A maximum clock drift rate of around 50 μs per second is observed.

Estimated time synchronization error is considered small for SHM applications, but clock drift is not negligible if long measurement records are taken. The delay of 10 μs corresponds to a 0.072-degree phase error for a mode at 20 Hz. Even at 100 Hz, the corresponding phase error is only 0.36 degree. However, after a 200 second measurement, the time synchronization error may become as large as 10 ms, corresponding to a 72-degree phase error for a 20 Hz mode. The resampling approach explained subsequently addresses this issue of clock drift.

4.2. Issues toward synchronized sensing

Accurate synchronization of local clocks on Imote2s does not guarantee that measured signals are synchronized. Measurement timing cannot necessarily be controlled accurately based on the global time. To allow better understanding on difficulties in realizing syncrhonized sensing, a brief explanation on the sampling process on the Imote2 is provided as follows. After the Imote2 triggers sensing, the sensor board proceeds to collect a predetermined number of data points and provide the data to the Imote2. Sensing time stamps for every N_{data} data points are available. The difficulties encountered in this process are explained next.

4.2.1. Uncertainty in start-up time

Starting sensing tasks at all of the Imote2 nodes in a synchronous manner under TinyOS is not possible. Even when the commands to start sensing are queued at exactly the same moment, the execution timing of the commands is different on each node. TinyOS is not a real time OS, and the waiting time before command execution cannot be deterministically predicted. Thus, the start of sensing will not be synchronized to each other.

In addition, the warm-up time for the sensing devices after the invocation of the start command is not deterministic. Even if the commands are invoked at the same time, sensing will not start simultaneously.

4.2.2. Difference in sampling rate among sensor nodes

The sampling frequency of the accelerometer on the available Imote2 sensor boards has nonnegligible random error. According to the data sheet of the accelerometer, the sampling frequency may differ from the nominal value by at most 10 percent (STMicroelectronics, 2009). Such variation was observed when 14 Imote2 sensor boards were calibrated on a shake table (see Fig. 4).

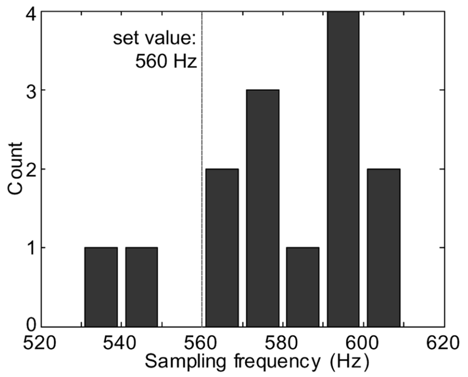

Fig. 4 Sampling frequencies of 14 sensor boards

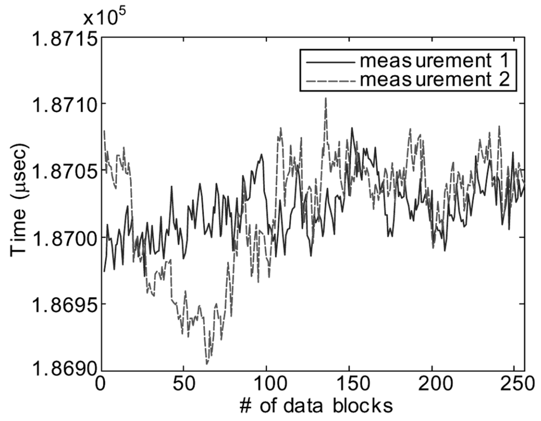

Fig. 5 Variation in the sampling frequency over time

4.2.3. Fluctuation in sampling frequency over time

Additionally, nonconstant sampling rate was observed with the Imote2 sensor boards, which if not addressed, results in a seriously degraded acceleration measurement. When a block of data is available from the accelerometer, the Imote2 receives the digital acceleration signal and obtain the time stamp of the last data point. By comparing differences between two consecutive time stamps, the sampling frequency of the accelerometer is estimated. Fig. 5 shows the difference between the timestamps when the block size is set at 110 data points. This figure provides an indication of the variation in the sampling frequency. The difference in two consecutive time stamps fluctuates by about 0.1 percent. Though imperfect time stamping on the Imote2 is a possible source of the apparent nonconstant sampling rate, the slowly fluctuating trend suggests the variable sampling frequency as a credible cause of the phenomenon. With this fluctuation, measurement signals may suffer from a large synchronization error and a nonconstant sampling rate.

4.3. Realization of synchronized sensing

Strict timing control is a possible solution for synchronized sensing. However, implementation of real-time operation makes the system large and complex. Real-time control of wireless sensors in a network is particularly challenging; the situation is exacerbated for the high sampling rates required in SHM applications. Also, even when a sensor node itself has real-time control, the peripheral devices such as the sensor chip may have execution time delay or uncertainty in timing.

Instead of pursuing real time control, synchronized sensing is realized herein using postprocessing on the non-real time OS of the Imote2, TinyOS. This post-processing approach depends less on the timing control quality of the sensors or timing properties of the smart sensor hardware; this approach should be easily adaptable to other systems and is considered a more universal approach. Note that even with a non-real time OS, the Imote2's high clock rate practically enables accurate timing control, as compared to real time systems with low clock rate.

Resampling based on the global time stamps addresses the three problems mentioned above. This section first reviews the basics of resampling and its polyphase implementation. The polyphase implementation is then modified to achieve finer resolution resampling. Finally, this proposed resampling method is combined with time stamps of measured data to address the three issues concurrently.

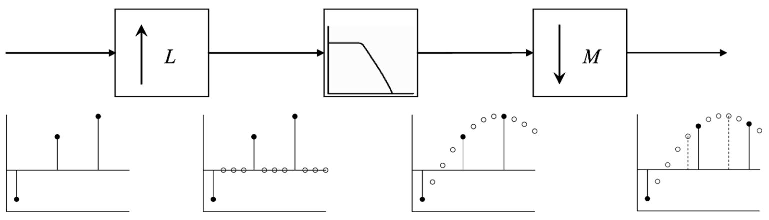

Fig. 6 Basic approach to resampling

4.3.1. Resampling

Resampling by a rational factor is performed by a combination of interpolation, filtering, and decimation (see Fig. 6). Consider the case in which the ratio of the sampling rate before and after resampling is expressed as an rational factor, *L/M*. The signal is first upsampled by a factor of *L*. The original signal *x*[*m*] is interpolated by *L*–1 zeroes as in Eq. (1),

$$v[l] = \begin{cases} x(m), & l = Lm, L = 0, \pm 1, \pm 2, \ldots \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where *v*[*l*] is the upsampled signal. A discrete-time low-pass filter with a cutoff frequency, π/L, is then applied to eliminate frequency components created by the insertion of zeroes. To scale properly, the gain of the filter is set to be *L*. Before decimation, all of the frequency components above the new Nyquist frequency need to be eliminated. A discrete-time lowpass filter with a cutoff frequency of π/M and gain of 1 is applied. This lowpass filter can be combined with the one in the upsampling process. The cutoff frequency of the filter is set to be the smaller value of π/L and π/M, and the gain is *L*. Decimation by a factor of *M* is then performed. Thus, the combination of upsampling, filtering, and decimation completes the resampling process.

One of the possible error sources of this resampling process is imperfect filtering. A perfect filter, which has a unity gain in the passband and a zero in the stopband, needs an infinite number of filter coefficients. With a finite number of filter coefficients, passband and stopband ripples cannot be zero. A filter design with 0.1 to 2 percent ripple is frequently used. Though this signal distortion during filtering is preferably suppressed, this process is not the only cause of such distortion. Other digital filters and AA filters also use imperfect filters. The filter in the resampling process needs to be designed so that it does not degrade signals as compared with the other filters.

4.3.2. Polyphase implementation

The polyphase implementation leverages knowledge that upsampling involves inserting many zeroes and that an FIR filter does not need to calculate the output at all of the upsampled data points. This implementation is explained in this section mainly in the time domain. Oppenheim, *et al.* (1999) provided a detailed description of the polyphase implementation in the Z-domain.

Upsampling and lowpass filtering with an FIR filter can be written in the following manner:

$$y[j] = \sum_{k=0}^{N-1} h[k]v[j-k]$$

$$v[l] = \begin{cases} x(m), & l = Lm, m = 0, \pm 1, \pm 2, \ldots \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $x[m]$ is the original signal, $v[l]$ is the upsampled signal, and $y[j]$ is the filtered signal. $h[k]$ is the filter coefficients for the lowpass FIR filter. The number of the filter coefficients is N. L is, again, an interpolation factor. By combining the two relationships in Eq. (2) and the following,

$$Lm = j - k \tag{3}$$

upsampling and filtering can be written as

$$y[j] = \sum_{m=\lceil (j-N+1)/L \rceil}^{\lfloor j/L \rfloor} h[j-Lm]x[m] \tag{4}$$

where $\lceil\ \rceil$ and $\lfloor\ \rfloor$ represent the ceiling and floor functions, respectively. The number of algebraic operations is reduced using the knowledge that $v[l]$ is zero at many points.

Downsampling by a factor of M is formulated as

$$\begin{aligned} z[n] &= y[nM] \\ &= \sum_{m=\lceil (nM-N+1)/L \rceil}^{\lfloor nM/L \rfloor} h[nM-Lm]x[m] \quad n = 0, \pm1, \pm2, \ldots \end{aligned} \tag{5}$$

where $z[j]$ is the downsampled signal. The outputs are calculated only at every M-th data point of the upsampled signal. This polyphase implementation, thus, reduces the number of numerical operations involved in resampling.

4.3.3. Fine resolution resampling utilizing initial delay compensation and interpolation

First, the resampling is generalized by introducing an initial delay, l_i. Eq. (5) is rewritten with l_i as follows:

$$\begin{aligned} z[n] &= y[nM + l_i] \\ &= \sum_{m=\lceil (nM+l_i-N+1)/L \rceil}^{\lfloor (nM+l_i)/L \rfloor} h[nM+l_i-Lm]x[m] \quad n = 0, \pm1, \pm2, \ldots \end{aligned} \tag{6}$$

where l_i is integer. By introducing l_i, the beginning point of the downsampled signal can be adjusted. When the time difference at the start of sensing is taken into account by l_i, the synchronization accuracy of signals is not limited by the original sampling period.

Precise frequency conversion through the polyphase implementation makes filter design quite difficult. For example, resampling of a signal from 100.01 to 150 Hz requires a lowpass filter with a cutoff frequency of ρ_l 15000. This filter needs tens of thousands of filter coefficients. Design of such a filter is computationally challenging. Also, a large number of filter coefficients may not fit in the available memory on smart sensors, and their use is computationally burdensome.

The combination of resampling and interpolation alleviates these filter design difficulties. The integer M is replaced by a real number M_r, and the integer l_i is replaced by a real number l_{ri}. The upsampling rate, L_a, remains an integer. Upsampling is the same as previously described. However, the downsampling process cannot be directly applied, due to the noninteger downsampling factor. Output data points to be calculated do not necessarily correspond to points on the upsampled signal. Output data points often fall between upsampled data points. Linear interpolation is used to calculate output values as follows:

$$\begin{aligned} z[n] &= y[p_l]\cdot(p_u-p_j)+y[p_u]\cdot(p_j-p_l) \\ &= \sum_{m=\lceil (p_l-N+1)/L_a\rceil}^{\lfloor p_l/L_a\rfloor} h[p_l-L_a m]x[m]\cdot(p_u-p_j) \\ &+ \sum_{m=\lceil (p_u-N+1)/L_a\rceil}^{\lfloor p_u/L_a\rfloor} h[p_u-L_a m]x[m]\cdot(p_j-p_l) \\ p_j &= nM_r+l_{ri} \\ p_l &= \lfloor nM_r+l_{ri}\rfloor \\ p_u &= p_l+1 \end{aligned} \tag{7}$$

A value of L_a ranging from 20 to 150 is employed in algorithmic testing and shown to yield reasonable results. While M_r and L_a are not uniquely determined, L_a needs to be in an appropriate range. A upsampling factor makes filter design challenging. On the other hand, a too small upsampling factor distort signals through interpolation.

4.3.4. Resampling-based synchronized sensing

The proposed fine-resolution resampling is employed to realize synchronized sensing. First, Imote2s start sensing nearly simultaneously without strict timing control. The Imote2s obtain a time stamp every N_{data} data points. The beginning of synchronized signals is specified in terms of global time. When the specified beginning time falls between the previous and the current time stamps, the delay l_{ri} is determined as the difference between the beginning time and the previous time stamp. By subtracting previous time stamp from the current one, the Imote2 estimates the sampling frequency and determines M_r. The fine resolution resampling is applied to this N_{data} points. When the next N_{data} points with the corresponding time stamp become available, M_r is determined in the same manner. l_{ri} is set to accounts for the time between the last resampled data point and the first data point of the current data set. The resampling process is applied and goes to the next N_{data} points. By repeating this process, synchronized sensing data is obtained.

5. Data aggregation

This section demonstrates that distribution and coordination can exploit application specific knowledge so that the data transfer requirements can be reduced without sacrificing performance of SHM algorithms. This data aggregation method is scalable to networks of large numbers of smart sensors (Nagayama, *et al.* 2006b, 2007a, Spencer and Nagayama 2006).

5.1. Analysis on the application model

In the analysis of dynamic response measurements of civil infrastructure where input force is unknown, correlation function based analyses are often employed. The Natural Excitation Technique (NExT; James, *et al.* 1993) states that correlation functions of vibration responses satisfy the equations of motion for free vibration. Correlation functions and their frequency domain representation, spectral density functions, are also used for more general purposes such as periodicity detection and time delay estimation.

Cross spectral density functions are, in practice, estimated from finite length records as in the following equation (Bendat & Piersol 2000):

$$\hat{G}_{xy}(\omega) = \frac{1}{n_d T}\sum_{i=1}^{n_d} X_i^*(\omega)Y_i(\omega) \tag{8}$$

where $\hat{G}_{xy}(\omega)$ is an estimate of cross spectral density function, $G_{xy}(\omega)$, between two stationary Gaussian random processes, $x(t)$ and $y(t)$. $X(\omega)$ and $Y(\omega)$ are the Fourier transforms of $x(t)$ and $y(t)$; * denotes the complex conjugate. T is time length of sample records, $x_i(t)$ and $y_i(t)$. Oftentimes, $x_i(t)$ and $y_i(t)$ are windowed and individual sets of signals may overlap in time. The normalized RMS error, $\varepsilon(|\hat{G}_{xy}(\omega)|)$, of the spectral density function estimation is given as

$$\varepsilon(|\hat{G}_{xy}(\omega)|) = \frac{1}{|\gamma_{xy}|\sqrt{n_d}} \tag{9}$$

$$\gamma^2_{xy}(\omega) = \frac{|G_{xy}(\omega)|^2}{G_{xy}(\omega)G_{yy}(\omega)} \tag{10}$$

where $\gamma^2_{xy}(\omega)$ is the coherence function between $x(t)$ and $y(t)$, indicating the linear dependence between the two signals. The random error is reduced by computing an ensemble average from n_d different or partially overlapped records. The estimated spectral densities can then be converted to correlation functions via the inverse Fourier transform.

5.2. Model based data aggregation

This study incorporates the correlation function estimation process into data aggregation to reduce data transfer requirements. The estimation process requires data from two sensor nodes. Measured data needs to be transmitted from one node to the other before data processing takes place. Associated data communication can be prohibitively large without careful consideration of the implementation. Two approaches for the estimation of these functions are explained next.

An implementation of correlation function estimation in a centralized data collection scheme is shown in Fig. 7, where node 1 works as a reference sensor. n_s nodes, including the reference node, are measuring structural responses. Each node acquires data and sends it to the reference node. The reference node calculates the spectral density as in Eq. (8). This procedure is repeated n_d times and averaged. For simplicity, no overlap between individual data sets is assumed. After averaging, the inverse Fourier transform is taken to calculate the correlation function. All calculations take place at the reference node. When the spectral densities are estimated from discrete time history records of length

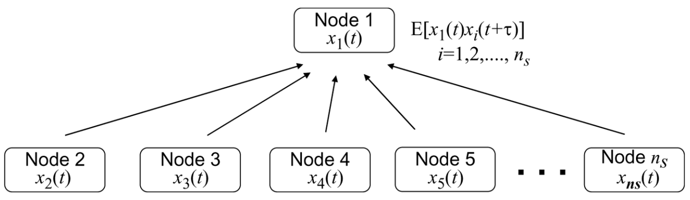

Fig. 7 Centralized correlation function estimation

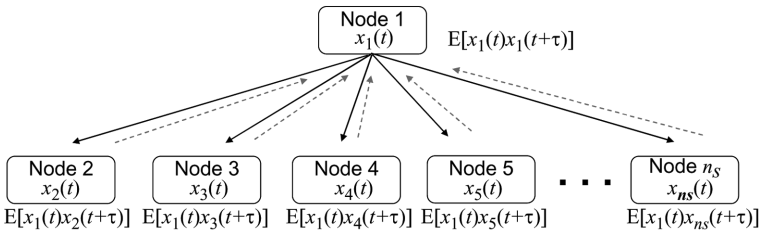

Fig. 8 Distributed correlation function estimation

N, the total data to be transmitted over the network using this approach is $N \times n_d \times (n_s - 1)$.

In the next scheme, data flow for correlation function estimation is examined and data transfer is reorganized to take advantage of computational capability on each smart sensor node. After the first measurement, the reference node broadcasts the time record to all of the nodes. On receiving the record, each node calculates the spectral density between its own data and the received record. This spectral density estimate is stored locally. The nodes repeat this procedure n_d times. After each measurement, the stored value is updated by taking a weighted average between the stored value and the current estimate. In this way, Eq. (8) is calculated on each node. Finally the inverse Fourier transform is taken of the spectral density estimate locally. The resultant correlation function is sent back to the reference node. Because subsequent modal analysis algorithms such as ERA uses, at most, half of the correlation function $N/2$ data points are sent back to the reference node from each node. The total data to be transmitted in this scheme is, therefore, $N \times n_d + N/2 \times (n_s - 1)$(see Fig. 8).

As the number of nodes increases, the advantage of the second scheme, in terms of communication requirements, becomes significant. The second approach requires data transfer of $O(N \cdot (n_d + n_s))$, while the first one needs to transmit data of the size of $O(N \cdot n_d \cdot n_s)$. The distributed implementation leverages knowledge regarding the application to reduce communication requirements as well as to utilize the CPU and memory in a network efficiently.

The data communication analysis above assumes that all the nodes are in single-hop range of the reference node. This assumption is not necessarily the case for a general SHM application. However, Gao (2005) proposed a DCS approach for SHM that supports this idea. Neighboring smart sensors in single-hop communication range form local sensor communities and perform SHM in the communities. In such applications, the assumption of nodes being within single-hop range of a reference node is reasonable.

Further consideration is necessary to accurately assess the efficacy of the distributed implementation. Power consumption of smart sensor networks is not simply proportional to the amount of data transmitted. Acknowledgment messages are also involved. The radio listening mode consumes power, even when no data is received. However, the size of the measured data is usually much larger than the size of the other messages to be sent and considered the primary factor in determining power consumption. Reduced data transfer requirements realized by the proposed model-based data aggregation algorithm will lead to decreased power consumption.

6. Experimental evaluation

The three middleware services are implemented on the Imote2 and their performance is experimentally evaluated. Six Imote2s are placed on the three-dimensional truss model located at the Smart Structure

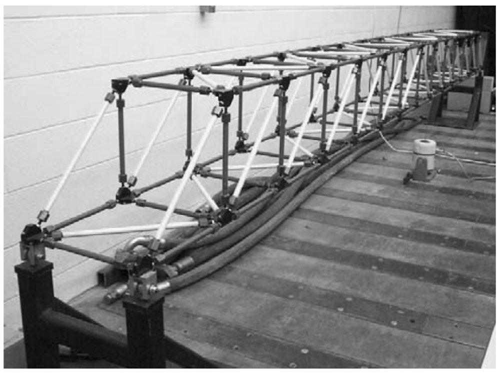

Fig. 9 Three-dimensional, 5.6 m-long truss structure

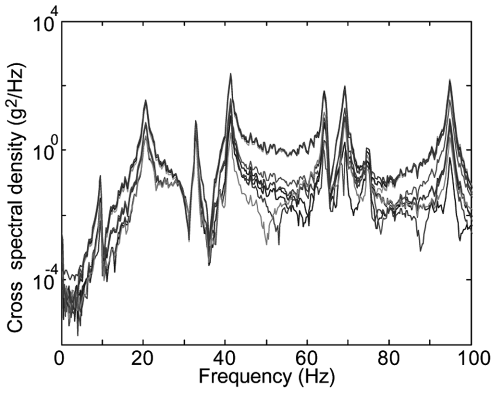

Fig. 10 Cross-spectral density estimates from Imote2 data

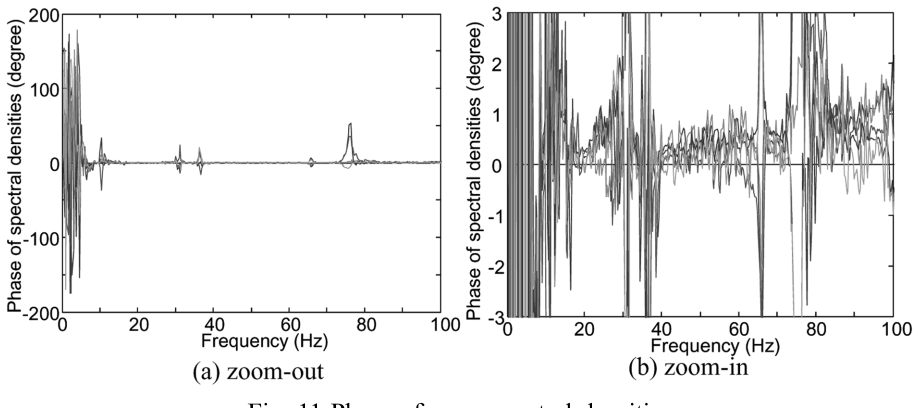

Fig. 11 Phase of cross spectral densities

Technology Laboratory of the University of Illinois at Urbana-Champaign (see Fig. 9; http://sstl.cee.uiuc.edu). These Imote2s are installed on the six structural nodes on one vertical outer plane of two adjacent truss bays and measure the acceleration response of this truss under band-limited white noise excitation, using the synchronized sensing middleware service. As explained in the model-based data aggregation section, the data is then transferred from a reference node to the other nodes using the multicast data transfer middleware service.

Correlation functions are then calculated and sent back to the reference and base station nodes using the reliable communication middleware service. Associated command transfer is accomplished by the unicast and multicast reliable command transfer middleware services.

The collected correlation functions are converted to spectral density functions and shown in Figs. 10 and 11. The amplitude of cross spectral density functions show clear peaks corresponding to structural vibration modes. The phase of the spectral density functions is considered to be constant over a wide frequency range if the sensors are installed on a rigid body and if signals are synchronized with each other. The slopes of the phase indicate synchronization errors. Though the truss does not behave as a rigid body, preliminary measurements with a wired system revealed that the observed portion of the truss does not have mode shape nodes in this frequency range; the slopes still indicate the synchronization errors. The flat phase in Fig. 11 demonstrates accurately synchronized sensing.

7. Summary

Middleware services for SHM applications using smart sensors have been developed. The data loss problem, which has adverse effects on an SHM algorithm, was addressed by developing reliable communication protocols. To realize synchronized sensing, a resampling-based approach was taken; the synchronized sensing enables highly accurate distributed sensing. Model-based data aggregation, including distribution of data processing and coordination among sensor nodes, provided scalability to a large number of smart sensors while preserving vibration signal information to be used in many of subsequent structural vibration analyses. These middleware services are implemented on the Imote2 and validated through a structural vibration measurement experiment. This development of the middleware services is expected to allow many SHM application users to obtain reliable structural response information from smart sensor networks smoothly and perform detailed structural analyses. These middleware services are open-source and available at: http://shm.cs.uiuc.edu/.

Acknowledgements

This work was supported in part by the National Science Foundation, under grant CMS 0600433 (Dr. S. C. Liu, program manager).

References

Bendat, J. S. and Piersol, A. G. (2000), *Random data: analysis and measurement procedures*, New York: Wiley.
Crossbow Technology, Inc. <http://www.xbow.com>
Elson, J., Girod, L. and Estrin, D. (2002), "Fine-grained network time synchronization using reference broadcasts", *Proceedings of 5th Symposium on Operating Systems Design and Implementation*, Boston, MA.
Ganeriwal, S., Kumar, R. and Srivastava, M. B. (2003), "Timing-sync protocol for sensor networks", *Proceedings of 1st International Conference On Embedded Networked Sensor Systems*, Los Angeles, CA., 138-149.
Gao, Y. (2005), "Structural health monitoring strategies for smart sensor networks", Doctoral dissertation, University of Illinois at Urbana-Champaign, 2005.
James, G. H., Carne, T. G. and Lauffer, J. P. (1993), "The natural excitation technique for modal parameter extraction from operating wind turbine", *Report No. SAND92-1666*, *UC-261*, Sandia National Laboratories, NM.

Kim, S., Pakzad, S., Culler D., Demmel, J., Fenves, G., Glaser, S. and Turon, M. (2007), "Health monitoring of civil infrastructures using wireless sensor networks", *Proceedings of the 6th international conference on Information processing in sensor networks*, Cambridge, MA, 254-263.

Lamport, L., Shostak, R. and Pease, M. (1982), "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems*, **4**(3), 382-401.

Lynch, J. P. and Loh, K. (2006), "A summary review of wireless sensors and sensor networks for structural health monitoring", *Shock Vib. Digest*, **38**(2), 91-128.

Maroti, M. Kusy, B., Simon, G. and Ledeczi, A. (2004). "The flooding time synchronization protocol", *Proceedings of 2nd International Conference On Embedded Networked Sensor Systems*, Baltimore, MD, 39-49.

Mechitov, K. A., Kim, W., Agha, G. A. and Nagayama, T. (2004), "High-frequency distributed sensing for structure monitoring", *Proceedings of 1st Int. Workshop on Networked Sensing Systems*, Tokyo, Japan, 101-105.

Nagayama, T., Rice, J. A. and Spencer, B. F., Jr. (2006a), "Efficacy of Intel's Imote2 wireless sensor platform for structural health monitoring applications", *Proceedings of Asia-Pacific Workshop on Structural Health Monitoring*, Yokohama, Japan.

Nagayama, T. Sim, S.-H., Miyamori, Y. and Spencer, B. F. Jr. (2007a), "Issues in structural health monitoring employing smart sensors", *Smart Struct. Sys.*, **3**(3), 299-320.

Nagayama, T. and Spencer, B. F. Jr. (2007), "Structural health monitoring using smart sensors", Newmark Structural Engineering Laboratory Report Series 001 http://hdl.handle.net/2142/ 3521.

Nagayama, T., Spencer, B. F., Jr., Agha, G. A. and Mechitov, K. A. (2006b), "Model-based data aggregation for structural monitoring employing smart sensors", *Proceedings of 3rd Int. Conference on Networked Sensing Systems (INSS 2006)*, Chicago, IL, 203-210.

Nagayama, T., Spencer, B. F. Jr. and Fujino, Y. (2007b), "Synchronized sensing toward structural health monitoring using smart sensors", *Proceedings of World Forum on Smart Materials and Smart Structures Technology* (SMSST 2007), Chongqing & Nanjing, China.

Oppenheim, A. V., Schafer, R. W. and Buck, J. R. (1999), *Discrete-Time Signal Processing*, Upper Saddle River, NJ: Prentice Hall.

Spencer, B. F., Jr. and Nagayama, T. (2006), "Smart sensor technology: A new paradigm for structural health monitoring", *Proceedings of Asia-Pacific Workshop on Structural health Monitoring*, Yokohama, Japan.

STMicroelectronics. <http://www.st.com/stonline/>.

MT