TinyOS-based real-time wireless data acquisition framework for structural health monitoring and control

Lauren E. Linderman¹, Kirill A. Mechitov² and Billie F. Spencer Jr.³,*†

¹Doctoral Candidate, Department of Civil and Environmental Engineering, University of Illinois, Urbana, IL 61801, USA
²Postdoctoral Research Associate, Department of Computer Science, University of Illinois, Urbana, IL 61801, USA
³Nathan M. Newmark Endowed Chair in Civil Engineering, University of Illinois, Urbana, IL 61801, USA

SUMMARY

Wireless smart sensor networks have become an attractive alternative to traditional wired sensor systems to reduce implementation costs of structural health monitoring systems. The onboard sensing, computation, and communication capabilities of smart wireless sensors have been successfully leveraged in numerous monitoring applications. However, the current data acquisition schemes, which completely acquire data remotely prior to processing, limit the applications of wireless smart sensors (e.g., for real-time visualization of the structural response). Although real-time data acquisition strategies have been explored, challenges of implementing high-throughput real-time data acquisition over larger network sizes still remain because of operating system limitations, tight timing requirements, sharing of transmission bandwidth, and unreliable wireless radio communication. This paper presents the implementation of real-time wireless data acquisition on the Imote2 platform. The challenges presented by hardware and software limitations are addressed in the application design. The framework is then expanded for high-throughput applications that necessitate larger networks sizes with higher sampling rates. Two approaches are implemented and evaluated on the basis of network size, associated sampling rate, and data delivery reliability. Ultimately, the communication and processing protocol allows for near-real-time sensing of 108 channels across 27 nodes with minimal data loss.

Copyright © 2012 John Wiley & Sons, Ltd.

Received 5 July 2011; Revised 13 April 2012; Accepted 16 June 2012

KEY WORDS: wireless sensor networks; smart sensors; data acquisition; communication protocol; real-time systems

1. INTRODUCTION

Civil infrastructure is essential for public safety and prosperity. The numerous bridge collapses, including the I-35 bridge collapse in Minnesota, highlight the importance of structural health monitoring (SHM) as a supplement to visual inspection. Furthermore, monitoring systems can allow engineers to evaluate a structural system after an extreme loading event such as an earthquake or typhoon. However, implementation of traditional wired monitoring systems can come at a high cost because of installation costs. In the literature, the average installed cost per channel of a wired monitoring system implemented in a building has been reported to be as much as $5000 [1], whereas for the 84 accelerometers deployed on the Bill Emerson Memorial Bridge, the cost was over $15,000 per channel [2]. Wireless smart sensors, which include onboard communication, processing, and memory, have the potential to significantly reduce these implementation costs and allow dense network deployments [3].

However, wireless sensor networks present inherent challenges to performing traditional monitoring. The limited network resources, including power and communication bandwidth, can make handling large quantities of data challenging [4,5]. Two common approaches to data acquisition in large sensor networks

*Correspondence to: Billie F. Spencer Jr., Nathan M. Newmark Endowed Chair in Civil Engineering, University of Illinois, Urbana, IL 61801, USA.
†E-mail: bfs@illinois.edu

Copyright © 2012 John Wiley & Sons, Ltd.
are used: data logging and decentralized data aggregation. In the first, data are acquired locally on sensor nodes prior to sending the measured data individually back to the base station. The collected time histories can then be analyzed. This data logging approach better utilizes the transmission bandwidth when compared with real-time acquisition; however, the process can take a significant amount of time. In the second approach, data are acquired locally and then processed, typically in small communities of neighboring sensor nodes; the aggregated data are returned to the gateway node [6]. This approach leverages the onboard computational power to reduce transmission size and power consumption [7]; however, complete time histories of the measured data are no longer available.

On the other hand, real-time data acquisition offers an alternative data collection approach, which can increase the applications of wireless sensors. For example, real-time acquisition allows wireless sensor networks to mimic tethered acquisition systems when real-time visualization of the response is desired. Furthermore, this approach may be desirable if actuation capabilities are included in the wireless system and real-time state knowledge is necessary (e.g., structural control). Specifically, real-time data acquisition implies that the sample is available within the sampling interval, whereas in near-real-time data acquisition, there is a latency of several sampling intervals in sample availability on the base station. Real-time data acquisition is required in applications such as wireless structural control. SHM applications are less sensitive to latency for real-time visualization of the response and thus near-real-time data acquisition suffices. Despite the onboard processing and communication capabilities, real-time data acquisition on wireless smart sensors is challenging because of operating system (OS) limitations, tight timing requirements, sharing of transmission bandwidth, and unreliable wireless radio communication.

Recent sensor systems have implemented real-time data acquisition by limiting network size, channels acquired, and/or sampling rates. Galbreath et al. [8] achieve continuous streaming on their own prototype sensor by acquiring three channels of 12-bit sensor data sampled at 1 kHz on a single sensor node. In this monitoring approach, multiple nodes were not required to communicate with the gateway node. Similarly, Paek et al. [9] limit the size of their networks and sampling rate to achieve sampling of 12 channels of acceleration across four nodes at 20 Hz using a TENET network with Stargate and MicaZ sensor nodes. Wang et al. [10] achieve reliable near-real-time transmission of 24 wireless sensors with 16-bit data at sampling rates up to 50 Hz on their own prototype sensor node. Their multithreaded operating system with multiple memory buffers does not require sending within one sample period and as a result can use a retry and acknowledgement protocol to ensure reliable communication. Niu et al. [11] achieve reliable near-real-time communication of 34 channels of 10-bit data across 17 nodes at up to 50 Hz on the IRIS sensor node by using a time-division multiple access (TDMA) protocol with a buffer of multiple samples. The TDMA protocol communication interval used is independent of the sampling rate or number of nodes and can result in buffer overflow at higher sampling rates. Whelan and Janoyan [12] achieve reliable real-time acquisition of 40 channels of 12-bit data over 20 nodes at a sampling rate of 128 Hz on the TmoteSky sensor node through low-level modification of TinyOS-1.x handling of events. To achieve reliable communication, they retransmit lost data, which can introduce some latency. Although the Whelan and Janoyan [12] system exhibits impressive performance, the time synchronization among nodes is only viable for several minutes, which limits the sensing interval. Thus, although real-time data acquisition has been implemented, high-throughput, near-real-time data acquisition over large networks for an extended sampling interval has not been realized.

This paper presents the implementation of high-throughput, real-time, wireless data acquisition on the Imote2, an advanced smart sensor platform used extensively today [13–16]. The implications of hardware and software limitations on the implementation of real-time sensing are discussed and then addressed in the communication protocol and application design for real-time data acquisition, which is evaluated for systems requiring real-time state knowledge. Because TinyOS is used on a majority of wireless sensor networks, including the Imote2, the limitations discussed, and solutions presented are TinyOS-based [17]. The initial application framework is then expanded to provide high-throughput, near-real-time wireless data acquisition for applications requiring a larger network size. Two near-real-time approaches are considered and evaluated based on their resulting network size, sampling rate, and data delivery reliability. The communication protocol used accounts for the number of nodes in the network as well as the sending and processing times to ultimately achieve sampling of 108 channels over 27 nodes at sampling rates up to 25 Hz. In addition, no low-level operating system modifications or large memory buffers are required to achieve this improved performance when compared with previous implementations.
2. WIRELESS SENSOR PLATFORM AND ASSOCIATED CHALLENGES

In this section, the wireless smart sensor platform employed for this research is presented, and the relevant hardware and software are discussed in detail. The challenges of hardware and software imposed on real-time wireless data acquisition are also outlined.

2.1. Hardware description

Although numerous wireless sensing units, both academic and commercially available, have been developed for SHM applications [3], the Imote2 was selected for this work. The Imote2, pictured in Figure 1(a), is well suited for data intensive SHM applications because of its variable processing speed, large onboard memory, and low-power radio. Its XScale PXA271 processor offers variable processing speeds to optimize power consumption and performance. The onboard memory consists of 32 MB of flash, 256 KB of static random-access memory, and 32 MB of synchronous dynamic random access memory. The Imote2 utilizes the popular CC2420 low-power radio that can be combined with an onboard or external antenna for wireless communication over the 2.4 GHz wireless band. The radio offers a theoretical maximum transfer speed of 250 Kbits/sec.

The Imote2 platform does not provide an onboard analog-to-digital converter, instead, it may interface with a user-selected sensor board over its basic connectors. The ISM400 sensor board was selected for this work among the commercially available sensor boards for the Imote2 because of its high-sensitivity accelerometers and high resolution analog-to-digital converter with user-selectable sensing parameters [18]. The sensor board, shown in Figure 1(b), consists of a three-axis accelerometer (ST Microelectronics LIS344ALH), temperature and humidity sensor (Sensirion SHT11), light sensor (TAOS 2561), and an external 16-bit analog input. The four-analog signals interface with a 16-bit analog-to-digital converter (QF4A512), which offers user-selectable anti-aliasing filters and sampling rates.

2.2. Embedded software

The embedded software is an essential component in the design of the real-time wireless data acquisition. This section will discuss the four main components of the software for application development: the operating system, software architecture, time synchronization, and sensing approach.

2.2.1. Operating system. The operating system popular with numerous embedded wireless sensor networks, TinyOS, is used on the Imote2 [3,17]. TinyOS (www.tinyos.net) is a component-based operating system written in the NesC language, a version of C for embedded systems, which has limited memory requirements. The open-source software differs from embedded real-time operating systems and traditional OS schedulers in its concurrency model. Tasks are executed in a first-in-first-out (FIFO) manner and run to completion. They can only be interrupted by hardware-triggered interrupt handlers, called events [19]. The inclusion of asynchronous interrupts allows the system to interact with real-time hardware. Thus, two main code execution contexts exist: a task posted to the end of a queue and an asynchronous interrupt handler. Code running in the interrupt context must be kept short, as nested handlers for the same interrupt are not supported.

Figure 1. (a) Imote2 (b) Imote2 and ISM400.
2.2.2. Software architecture. Similar to the component-based operating system, the Illinois SHM Project Services Toolsuite (http://shm.cs.uiuc.edu/software.html) used in the development of real-time wireless sensing utilizes a modular service-oriented architecture. The framework consists of three main elements: foundation services, application (domain-specific) services, and tools and utilities [6]. A typical application would combine several foundation and application services. Several of the key foundation services to support real-time sensing include reliable communication and synchronized sensing. The reliable communication service allows reliable sends of different message types, including commands and long data sets. The synchronized sensing service combines time synchronization, which provides global timestamps, and resampling to account for sampling offset and variation of sampling rates [20].

2.2.3. Time synchronization. Precise time synchronization serves two key purposes in real-time sensing: (i) providing consistent global timestamps for synchronizing the data acquired from different sensor nodes and (ii) scheduling communication. Although approximately 1 ms precision typically suffices for communication scheduling, much tighter precision is needed for acquiring high-quality synchronized data. A custom time synchronization protocol for SHM applications on the Imote2 has been implemented [21]. With the extension of the Flooding Time Synchronization Protocol with clock drift estimation and compensation features, it maintains synchronization error within 80 μs over a period of several minutes without resynchronization.

2.2.4. Sensing approach. In general, the sensing application on the Imote2 interfaces with the sensor board through driver commands. The user first specifies the desired channels, sampling rate, and number of samples. The application relays this information to the driver when posting a sensing task. When the driver is initialized, sensing begins and the data are passed to the application through a buffer. Sensing continues until the desired amount of data has been acquired.

2.3. Implications of hardware and embedded software on implementation of real-time sensing

The TinyOS operating system design, although useful for embedded applications, makes the real-time scheduling and control required for real-time wireless data acquisition challenging. This section will outline how the event-driven concurrency model of TinyOS along with standard hardware limitations impacts real-time sensing.

2.3.1. Sampling rate limitation. The FIFO task queue and lack of priority-based scheduling limit the sampling rates possible for real-time data acquisition. Each data sample is passed through to the application from the driver in an event generated by an interrupt handler, which is similar to posting a task. Any processing tasks including calculating the global time stamps, temperature correction, and sending must occur before the next data sample is passed. Otherwise, the task queue will slowly fill, and the real-time nature is lost. Thus, the sample interval is limited by the total time required to process and send.

2.3.2. Communication time. To improve communication reliability, the radio utilizes a clear channel assessment to ensure that the wireless channel is free prior to transmitting. Thus, multiple nodes transmitting at the same time can increase communication time. Furthermore, because the radio waits a random back-off time prior to reassessing the channel, the time required to send while multiple nodes are transmitting is not consistent. Therefore, predicting the sending time, which is important for determining the sampling rate as mentioned previously, is challenging.

2.3.3. Sensing offset. The sensing approach, as well as variation in hardware start-up times, introduces an offset between the desired and actual start of sensing. A desired sensing start time is specified when the sensing task is posted; however, sensing does not begin at this exact specified time. Nagayama et al. [20] explain that although performing all sensing and timestamping within the interrupt context could be used to gain more accurate timing than posting a task, firing an interrupt with a computationally intensive interrupt handler (transferring data over the serial peripheral interface bus, decoding and storing it, and adding a timestamp) at a high frequency is unreasonable in the TinyOS concurrency model. Furthermore, variation in hardware initialization times would result in a delay nonetheless.
As a result, the sensing approach, illustrated in Figure 2, accepts relative uncertainty in the start time for sensing. When the driver initializes, sensing begins; however, samples are not stored and passed to the application until they are within a sampling interval of the desired start time, $t_{\text{start}}$. This offset is non-trivial and, because of variation in processing of the sensing task and the hardware initialization time, is non-deterministic. In local data logging approaches, this sensing offset is recorded and accounted for during post-processing by resampling the data prior to transmission [21].

However, the strict timing of real-time transmission requires accounting for this offset during sensing in the application design. Although time synchronization aligns the global clocks among the nodes, the sampling times are not consistent because of this offset. Thus, any scheduling among nodes based on sample ready events will not be aligned. Furthermore, the time stamps of the data must be transmitted as well, so the offset can be accounted for later in resampling, if desired.

3. REAL-TIME DATA ACQUISITION SERVICE FOR REAL-TIME STATE KNOWLEDGE

The sampling rate limit, sensing approach, and communication latency limitations due to the design of wireless sensor hardware and TinyOS described in the previous section must be addressed in the application design. Consequently, unlike wired systems, implementation of real-time wireless data acquisition requires addressing the tradeoff between performance, including network size and sampling rate, and reliability. The resulting wireless data acquisition service, which could be applied to structural control or health monitoring applications, and its performance, will be presented in this section.

3.1. Application design

Given the FIFO scheduling of TinyOS, minimizing the time for each element of a sampling interval and providing a consistent time to send are necessary for determining the maximum sample rate possible. Because of the random communication latency when multiple nodes send at the same time, a scheduled communication approach is utilized. Furthermore, within this framework, the amount of data returned to the gateway nodes is minimized to the 8-bit node ID, four channels of 16-bit data, and a 32-bit timestamp for accurate reconstruction of the data. Thus, the total packet payload is limited to a minimum of 14 bytes.

3.1.1. Communication protocol. The common TDMA protocol is implemented to allow multiple leaf nodes to communicate with a single receiver or gateway node, by transmitting in different time slots. A TDMA protocol, illustrated in Figure 3, permits only one node to send at a time, thus allowing the communication time to be more readily determined because of the absence of contention and back-off delays. The complete sensing interval is broken down into three components: processing on the leaf node (or remote processing), send time, and processing on the local (or gateway) node. The processing on the leaf node includes any handling of the sensor data and timestamping. The send time consists of the time from calling sent on the remote node until the completion of packet transmission. Finally, processing on the local node consists of extracting and decoding the data from the packet on the gateway node and transferring it to an internal buffer.

Because a reliable communication protocol involving acknowledgements and resends may take an undetermined amount of time, a generic or unreliable communication scheme with only a cyclic-redundancy check for packet error detection is used. Thus, if bit errors are found within the packet, the data packet is dropped and no retransmission occurs. Although this GenericComm scheme does not address packet loss, a relatively consistent send time is possible [21]. Furthermore, a TDMA protocol reduces the loss of packets because of collisions by limiting the likelihood of multiple nodes

Figure 2. Sensing approach.
transmitting at the same time. Although collisions are only one of numerous causes of packet loss, including path loss and antenna orientation, a TDMA protocol can help to improve reliability [22].

Because of hardware variations among Imote2s and the event-driven nature of TinyOS, the time required for sending and processing will vary both among sensor nodes, as well as on an individual node. Thus, a timing analysis was conducted on several sensor nodes to assess the time required for each step in a single sampling interval; remote processing, send time, and processing on the gateway node are recorded for 500 samples over four trials for several nodes. A cumulative distribution function of the discrete results was calculated for each step, as shown in Figure 4, and the 97th percentile values were selected to ensure that each step in the sampling interval typically occurred within the time allotted. The timing analysis results are given in Table 1. For the processing steps, the variation in time required to complete the tasks is small; however, the variation in sending time is significant. Thus, although selecting the 97th percentile value decreases the maximum possible sampling rate, the larger send time will improve reliability by guaranteeing that most sends will occur within the time allotted. Ultimately, the combination of the processing and sending times on the leaf node are used to calculate the delay to be employed in the TDMA scheme.

In addition, the variable processing speed of the Imote2 is utilized to reduce the time required for each step. The speed is increased from the normal operating speed of 13–104 MHz to achieve this performance. Because the processing time given in Table 1 is so much smaller than the sending time,
the processing speed is not increased to the maximum of 416 MHz because of the significantly greater power consumption at higher speeds [23,24].

The variations in the cumulative distribution functions (Figure 4) also highlight the effects of underlying processes on the sensor nodes. The staircase pattern of the empirical CDF of the send time can be attributed to the random back-off time in the wireless communication protocol to limit collisions and bandwidth contention. The discontinuities in the processing CDFs are likely due to background tasks executed prior to the completion of packet transmission by the non-preemptive FIFO scheduler used in TinyOS. Because both radio and memory management tasks are included in the gateway processing, the time variation is larger.

3.1.2. Sensing offset. The TDMA communication protocol assumes that all nodes are sensing at the same time; however, as discussed in Section 2, there is an offset in the exact time of sensing for each node. This offset, which is not known prior to the start of sampling, must be accounted for in the communication scheduling to ensure that sends do not overlap despite using a TDMA approach. Furthermore, the time stamp must be returned with the data to account for this offset in resampling, which increases the packet payload for each sample.

3.2. Application flowchart

The complete application requires combining accurate time synchronization and reliably sent commands to start sensing with this scheduled communication approach. Figure 5 illustrates the combination of these services into the overall program flow. At the start of the application, the user inputs the sensing parameters including the channels, number of samples, sampling rate, and leaf nodes for which data are to be acquired. These parameters are sent to the leaf nodes reliably to initialize the application. Time synchronization then occurs to ensure the leaf node’s clocks are aligned, which is necessary to provide reasonable alignment in sensing and allow the tight scheduling of sends in the TDMA protocol. After synchronization, a message for calculating the appropriate delay in sending for the communication protocol is sent reliably to the responsive nodes. The two initialization messages are sent reliably because they are essential to successful completion of the application, and as such, more time is allotted for these messages. Once sensing begins, the continuous sensing and sending protocol starts and continues until the leaf nodes have acquired and sent all the desired number of samples.

<table>
<thead>
<tr>
<th>Timing analysis for steps in wireless data acquisition.</th>
<th>Processing leaf (ms)</th>
<th>Sending leaf (ms)</th>
<th>Processing gateway (ms)</th>
<th>Total time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>97th percentile</td>
<td>0.50</td>
<td>6.55</td>
<td>1.5</td>
<td>8.55</td>
</tr>
<tr>
<td>Mean</td>
<td>0.37</td>
<td>4.17</td>
<td>1.27</td>
<td>5.81</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.09</td>
<td>1.48</td>
<td>0.20</td>
<td>1.77</td>
</tr>
</tbody>
</table>

Figure 5. Overall application flowchart.
Because the continuous sampling component is the central part of real-time wireless data acquisition, it is presented in more detail in Figure 6. When a sample is passed from the sensor board driver to the application, a sample ready event is signaled. Next, the time for a send timer interrupt is calculated on the basis of the time the sample is received, the start of sensing offset, and the sending delay determined for the TDMA scheme. If the time calculated is greater than one sampling interval because of the sensing offset, then it must be accounted for when setting the timer interrupt and determining the appropriate packet to send when the interrupt fires. A timer interrupt is used to signal the send rather than the default TinyOS timer that uses task posting, as accurate scheduling is required for the TDMA scheme. Once the interrupt is set, the sample is processed. The time stamp is calculated, and temperature correction of the acceleration data is applied if necessary. After the interrupt is fired, the selected radio packet is sent unreliably to the gateway node.

The interrupt time is calculated and set prior to processing the data because the time required for processing has some variation, as mentioned previously; as such, the time for processing is encompassed in the delay for the TDMA scheme. Thus, the total sending delay determined for the TDMA scheme is based on how many nodes are there in the system and the total time required to process and send the packet. Including the number of nodes and the sending/processing times in the TDMA approach makes this approach unique from other medium access control layer protocols, which cannot account for these variables [25,26].

3.3. Application performance

Given the application design and timing analysis, the resulting performance of real-time wireless data acquisition in terms of network size, sampling rate, and throughput, when only sensor data are considered, is provided in Table 2. Because of the TDMA scheme, the maximum sampling rate decreases as the number of sending nodes in the network increases. However, the maximum data throughput stays relatively unchanged because of the increase in network size.

The maximum data throughput is lower than the theoretical maximum available on the radio band because of the TDMA approach and FIFO task scheduling of TinyOS. For example, if the entire radio packet is used during transmission, including the preamble, headers, maximum data size, and footer as shown in Figure 7, the maximum data throughput achievable using a timeslot length of 7.1 ms for this TDMA scheme is about 149 Kbps. However, if only the maximum possible data payload is considered

![Figure 6. Continuous sensing flowchart for leaf node.](image)

### Table II. Performance of real-time wireless data acquisition.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Sampling rate (Hz)</th>
<th>Max. data throughput (Kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>115</td>
<td>7.36</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>7.68</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>7.68</td>
</tr>
</tbody>
</table>
in the throughput calculations, the maximum data throughput possible further reduces to about 125 Kbps. In our specific application, because of the small payload size, the data throughput is significantly lowered from the maximum possible. Therefore, as the TDMA scheme and scheduling communication around sensing offer a solution, it is at the cost of significant performance.

4. HIGH-THROUGHPUT NEAR-REAL-TIME WIRELESS DATA ACQUISITION

For applications that only require near-real-time sensing, such as SHM, the performance of real-time wireless data acquisition discussed in Section 3 can be significantly improved by buffering samples. The performance improvement is seen in the network size and associated sampling rate and data throughput. However, there is a tradeoff between the latency, network size, and sampling rate because they are directly related to the number of samples buffered prior to sending. As such, the design and performance of two different buffering sizes are presented: 3-sample buffer and a 9-sample buffer.

4.1. Application design

The application design for a buffered approach mirrors the design for real-time data acquisition presented in Section 3. A scheduled communication approach is still used; however, it is expanded to utilize the advantage of buffering of multiple samples within one packet prior to sending. Within this framework, the data returned to the gateway include the desired number of buffered samples, which comprised four channels of 16-bit sensor data, an associated 32-bit time stamp, and an 8-bit node ID. Thus, the payload when buffering three and nine samples is 38 and 110 bytes, respectively. A maximum of nine buffered samples is considered, because the maximum data payload of one radio packet dictated by the IEEE 802.15.4 protocol and TinyOS 1.x standard medium access control protocol is 112 bytes (Figure 7). The three sample buffer offers an increase in network size over the previous approach with a relatively small increase in payload size, which will slightly decrease the maximum sampling rate as discussed later.

4.1.1. Communication protocol. Similar to the previous design, a scheduled TDMA communication protocol is used to allow multiple leaf nodes to communicate with one gateway node in a consistent and more reliable manner. However, buffering of multiple samples prior to sending allows the number of nodes in the network to increase for a comparable sampling rate. As shown in Figure 8, a staggered TDMA approach is used on the basis of the number of samples buffered. For example, when three samples are buffered, three sampling intervals can be used for sending. Thus, the TDMA approach illustrated in Figure 3 can be applied to all three sampling intervals.

Similar to before, a timing analysis was conducted on several sensor nodes, in which the time for each step was determined for 250 samples over nine trials for several nodes. A cumulative distribution function of the discrete results was calculated for each step, and the 97th percentile values were selected for reliability. The timing analysis results for both approaches are given in Table 3.

Although the buffered approach allows the size of the network to increase, the resulting sample rate will decrease because of the additional time required for each step of a sampling interval: remote processing, remote send, and local processing. The difference in processing time on the remote node is negligible for the different approaches because the sample processing is the same. However, the sending time increased for the 3-sample buffer and again for the 9-sample buffer because of the larger packet payloads. This increase will have the most significant impact on the maximum sampling rates possible.
4.2. Application framework

Similar to real-time data acquisition, the near-real-time approach requires tight time synchronization and reliable commands to start sensing in combination with the staggered communication protocol. Thus, the general application flowchart matches that presented in Figure 5. The main difference in the approaches is the calculation and setting of the send timer interrupt. Because samples are buffered, the send interrupt is only set every \(n\) samples when an \(n\)-sample buffer is used. Furthermore, although the interrupt is based on the same calculation of current time, sensing offset, and TDMA send delay, accounting for this calculated time being greater than one sample period is made simpler by the buffering of multiple samples.

4.3. Application performance

Near-real-time wireless data acquisition can significantly improve performance in terms of network size and associated maximum sampling rate and throughput; however, there is a tradeoff between the network size, sampling rate, and latency because of the number of samples buffered. Furthermore, an increase in the number of samples buffered means a higher number of samples will be lost if a packet is dropped using unreliable communication. The resulting performance of the application in both sampling rate and reliability is presented in this section.

4.3.1. Sampling rate and throughput. Given the timing analysis presented in Table 3 and the application design, the resulting performance in terms of network size and associated maximum sampling rate and throughput is presented in Tables 4 and 5. The resulting network size and data throughput are significantly improved over the previous approach by buffering samples. Furthermore, the drop in the maximum possible sampling rate for the network is not significant considering the large increase in network size. This large increase in network size and associated packet payload is the biggest contributor to the increase in data throughput.

Table III. Timing analysis for time-division multiple access approach with buffered samples—97th percentile values.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Processing leaf (ms)</th>
<th>Sending leaf (ms)</th>
<th>Processing gateway (ms)</th>
<th>Total time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-sample</td>
<td>0.4</td>
<td>7.7</td>
<td>2.3</td>
<td>10.4</td>
</tr>
<tr>
<td>9-sample</td>
<td>0.4</td>
<td>10.1</td>
<td>2.0</td>
<td>12.5</td>
</tr>
</tbody>
</table>

Table IV. Application performance for 3-sample buffer approach.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Sampling rate (Hz)</th>
<th>Max. data throughput (Kbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–3</td>
<td>100</td>
<td>19.2</td>
</tr>
<tr>
<td>4–6</td>
<td>50</td>
<td>19.2</td>
</tr>
<tr>
<td>7–9</td>
<td>35</td>
<td>20</td>
</tr>
</tbody>
</table>

Figure 8. Staggered time-division multiple access protocol for 3-sample approach with 6 leaf nodes.
4.3.2. Data delivery performance. Because an unreliable communication protocol is used in combination with a timed communication scheme, some data loss is expected. However, the packet loss due to the application design and chosen sending delays is expected to be minimal. Because multiple samples are buffered into one packet, a lost packet corresponds to more lost data and thus is a greater concern and needs to be investigated.

To determine the data delivery performance of near-real-time data acquisition application, the application was evaluated in a near perfect communication environment. The sensor nodes with a mix of onboard and external antennas were placed evenly, spaced in an open environment with a clear line-of-sight to the gateway node. The 3-sample approach was tested in an outdoor parking garage on the University of Illinois campus as pictured in Figure 9(a). Because of inclement weather, the 9-sample approach was conducted in a classroom in the Newmark Civil Engineering building on the university campus as shown in Figure 9(b). Five trials of continuous data acquisition of several hundred samples at key sample rates for each approach and network size were conducted. Two different node configurations for each network size were considered. The complete testing matrix is provided in Table 6. Fewer samples were taken in each trial of the 3-sample approach to prolong battery life over the tests, whereas, the nodes in the 9-sample approach were powered with USB.

The data delivery results for each approach are given in Figures 10 and 11, respectively. The average reception rate gives an indication of data loss because it accounts for data sample loss, not packet loss. In general, minimal data loss was observed. The average reception rate for the 9-sample approach was slightly lower than the 3-sample approach, which is expected as each dropped packet contains more samples; however, the average reception rate is higher than 97%, which was the selection cutoff for timing parameters.

![Figure 9](image-url)
In addition to average reception rate, the maximum cluster of samples dropped was calculated. The maximum cluster size gives an indication of burst loss, which is more of a concern when samples are buffered. Furthermore, a small cluster size illustrates that the application is able to recover if the timed
scheme fails for a sample and that the transmission errors do not accumulate. For the 3-sample approach, typically only one packet is dropped. The maximum cluster size indicates about two packets are dropped for the 9-sample approach. The slightly poorer performance could be accounted for by the indoor testing environment, which has a higher likelihood of poor communication due to multi-path effects and other wireless networks or devices operating locally on the 2.4-GHz band. In general, however, the maximum cluster size is small for both approaches.

Overall, these results highlight the tradeoff between the number of samples buffered, network size, maximum available sampling rate, and reliability. The 9-sample approach significantly increases the network size for a small increase in sampling interval; however, the average reception rate is lower, because a lost packet corresponds to more data loss. Thus, the real-time data acquisition application can be tailored on the basis of the desired network performance; that is, for minimum latency, the unbuffered approach is used, for maximum network size with high sampling rates, the 9-sample buffer is used, and for balanced throughput, latency, and reliability, the 3-sample buffer is used.

5. CONCLUSION

This paper presents the implementation of high-throughput real-time wireless data acquisition on the Imote2 platform. Although this implementation is specific to the Imote2, the hardware and software challenges addressed are common to many available platforms. The resulting application framework for real-time data acquisition and its performance are presented. The application is expanded for high-throughput applications that require large network sizes and high sampling rates. Ultimately, the communication and processing protocols allow for near-real-time sensing of 108 channels across 27 nodes at up to 25 Hz with minimal data loss and require no low-level modifications to the operating system or large memory buffers.

The event-driven nature of TinyOS, communication latency, and existing sensing framework necessitate a tightly scheduled approach to achieve real-time data acquisition. Accurate time synchronization, reliable initialization commands to start sensing, and TDMA communication protocol are combined to achieve wireless real-time data acquisition. By buffering samples, this application framework is expanded to increase network size and throughput while maintaining high sampling rates. Because a tradeoff exists between the number of samples buffered, latency, network size, sampling rate, and reliability, two buffer sizes are considered: 3 and 9-sample buffers. The network size, associated sampling rate and throughput, and data delivery performance are investigated for both buffer sizes. Both approaches, particularly the 9-sample approach, increase the network size for a relatively small increase in sampling interval. Thus, high-throughput near-real-time wireless data acquisition that is viable over an extended period is successfully implemented on the Imote2 smart sensor platform. Furthermore, the appropriate real-time data acquisition service can be selected from the three approaches, including the unbuffered, 3-sample buffer or 9-sample buffer, on the basis of the network goals, that is, low latency for real-time control or large network size, and throughput for monitoring applications.

The application framework allows for future improvements, including network size and reliability. A frequency-division multiple access approach could be utilized alongside the current design to have multiple smaller networks operating on different radio bands for a larger total network size. Finally, the data could be logged locally on the leaf nodes and retransmitted later if completely reliable data acquisition is necessary.

Contract/Grant: National Science Foundation; contract/grant numbers: CMMI-0928886 CNS-1035773

REFERENCES


