Design Of Smart Sensing Component For Volcano Monitoring

Yang Peng∗ Richard Lahusen† Behrooz Shirazi∗ WenZhan Song∗

∗Sensorweb Research Laboratory, Washington State University, Vancouver, WA 98686, USA.
Email: yang_peng@wsu.edu songwz@wsu.edu shirazi@wsu.edu
†Cascades Volcano Observatory, U.S. Geological Survey, Vancouver, WA, USA.
Email: rlahusen@usgs.gov

Keywords: wireless sensor networks, smart sensing, high frequency, high fidelity, real-time.

Abstract

With the growing number of Wireless Sensor Network (WSN) monitoring systems in outdoor environments, smart sensing capability has attracted great research interests recently: scientists and researchers expect long-term healthy systems running in harsh environments. Because of the environmental damages to less-protected sensors, a sensing component, highly desirable to be robust and faithful for high data fidelity. In this paper, we propose a smart sensing design solution aimed to help acquire real-time, faithful scientific monitoring data from the active volcano, Mt. St. Helens. Our smart sensing design provides services of online-configurable sensor driver, fault detection, data prioritization and time synchronization. It could be applied to sensing component design in multiple wireless sensor network applications. This design solution has been implemented on a test-bed of sixteen OASIS [1] prototype sensor nodes.

1 Introduction

In the last few years, many wireless sensor network systems have been developed for environment monitoring. The sensing component in WSN systems such as structure monitoring [13], ocean observation [6], volcano monitoring [12] and etc. plays an important role in the system architecture. The sensing data itself is the most valuable information inside the network, and it should obtain the same attention as routing, medium access control (MAC) or other components. A robust and elaborate sensing component would help in alleviating the harm to the system caused by broken sensors, producing high fidelity scientific data, reducing insignificant information transmission and reporting emergency events on time. Thus the smart sensing design could improve system performance and prolong the system’s lifetime in harsh environments. In order to satisfy the requirement of robustness, faithfulness and timeliness of the volcano monitoring system from USGS and NASA JPL, we designed the smart sensing component for the in-situ network of the OASIS project. This component is designed to collect real-time high frequency monitoring data from the active volcano - Mt. St. Helens.

During the development, we met several challenges. First, the high frequency sampling (up to 1KHz) of multiple sensors which is always required for monitoring systems, posts a great challenge for resource constrained sensor nodes. In the OASIS project, each sensor node has four sensor channels: both seismic and infrasonic sensor samples 200 bytes data per second, the lightning sensor samples in 10Hz rate with 16-bit resolution, and the GPS receiver produces about 200 bytes data every ten seconds. The raw data rate is about 3.5Kbps in total. Under such a high data rate, the demands of large storage space and high CPU processing capability increase design difficulties.

The second problem we met is how to guarantee highly accurate global time synchronization. Currently, there are many time synchronization protocols developed for wireless sensor networks and they achieved good research results. With the help of these protocols, sensor nodes in the network could stamp the sensing data with a finer grain of real-time information. Most of the protocols need to connect a GPS receiver to the sink node to build a precious global time root of the network. However, such a mechanism is not robust enough in harsh environments, and many factors would lower the time accuracy, such as damage to the GPS receiver, GPS signal disappearing due to the location of antenna and the drift of the free running real time clock. What’s more, for volcano monitoring applications, seismologists expect synchronized sampling of each station in the network to locate the spot and time of the earthquake event and predict earthquake movements. The sampling time differences among the network should be limited to one millisecond. Such an expectation even raises the design requirement of the time synchronization component in the OASIS system.

Another challenge we are facing is the harsh volcano environment. The active volcano environment increases the possibility of negative affection to the sensing component than other components. Sensors, exposed to the volcano surface, are easier to be damaged, and it is difficult to replace the broken part on-site. The sensor node continuously running with a broken component will impact the whole network.

After carefully considering the challenges mentioned above, we propose the Smart Sensing Design Solution to guarantee robustness, faithfulness and timeliness of data collection in volcano monitoring. The rest of the paper is organized as follows. In Section 2, we review the re-
lated works to sensing component design. In Section 3, we present the design details of our smart sensing component. Finally, the conclusions are presented in Section 4.

2 Related Works

2.1 Fault Detection

[11] developed an online detection of sensor faults technique by comparing the results of multisensor fusion with, and without, each of the sensors involved using non-linear function minimization and then identifying the faulty sensor using non-parametric statistical techniques. Their simulation results indicate the high accuracy of the approach, but the implementation complexity of non-linear function minimization is too high for resource constrained sensor nodes. [2] proposed a localized faulty sensor identification algorithm which requires low computational overhead and is easily scaled to large sensor networks. The reading at a sensor is compared with its neighbors’ median reading and the sensor is very likely to be faulty if the absolute difference is large.

2.2 Event Detection

In [17], the SAX [7] stream data mining method is used to detect complex events in wireless sensor networks. This approach transforms the time-stream sampling data into a symbolic representation and declares an event based on the distance metrics. There are three main phases in their detection algorithm: 1) the learning phase where the algorithm keeps track of the distances it has seen; 2) in the Initial Detection Phase, the SAX is called to convert adjacent sets of samplings to strings and calculate their distance, if the distance between two strings is greater than the maximum distance observed during learning, an event is reported; 3) the Escalation Phase could vary the window size over the samplings and produce progressively larger strings to increase the accuracy of the algorithm. However, the requirement of floating point calculation and storage space in their approach is not feasible for high data rate WSN systems.

The system in [10] reports alarms and events that are judged to be of high priority by a simple rule which is based on a decision engine of spatio-temporal cross-correlation of the available sensor inputs. In the initial trial, various alarm conditions were programmed into each node in the forms of rules. The rules were checked each hour when the logger was read, and if a condition was satisfied an alarm code was generated. This alarm code was then propagated to the base station. Our implementation takes this idea of alarm report. In our system we provide the configurable alarm report mechanism instead of using the correlation information from neighboring nodes.

In [14], the author proposed an event detection mechanism based on matching the contour maps of in-network sensory data distribution. Their approach for event detection is also based on the spatio-temporal pattern in sensor readings instead of simple thresholds: the changes in the sensor readings of networked nodes that are caused by an event usually exhibit some spatio-temporal pattern. Their system needs to build a contour map of the network, and the event query is generated by the snapshot of the contour map in a user defined time period. If the user query pattern matches the snapshot with a confidence level, an event is reported. Such a event detection approach is useful in querying the network, however, in our system, our event detection processing is aimed to mark high priority event data and reduce the data transmission based on the real-time analyzing result.

2.3 Time Synchronization

In RBS [4], a reference message is broadcasted. The receivers record their local time when receiving the reference broadcast and exchange the recorded times with each other. RBS eliminates transmitter-side non-determinism but additional message exchange is necessary to communicate the local time-stamps between the nodes. The TPSN [5] algorithm performs pairwise synchronization along the edges of a spanning tree of the network. Each node gets synchronized by exchanging two synchronization messages with its reference node one level higher in the hierarchy. The TPSN achieves a better performance by time-stamping the radio messages in the Medium Access Control layer of the radio stack and by relying on a two-way message exchange. However, TPSN does not estimate the clock drift of nodes, which limits its accuracy, and does not handle dynamic topology changes.

The Flooding Time Synchronization Protocol - FTSP [9] has been successfully integrated into some existing WSN systems [12]. FTSP is robust against node and link failures and it achieves the robustness by utilizing periodic flooding of synchronization messages and implicit dynamic topology updates. By utilizing MAC-layer times-tamping and comprehensive error compensation including clock skew estimation, FTSP reached a high precision performance. In FTSP, the flooding messages take low communication bandwidth. Though FTSP achieves a high global time accuracy across the network, damages to the GPS receiver (loss of time root) could affect the whole time synchronization component. The OASIS system provides multiple GPS receiver across the network and provides a robust time synchronization mechanism.

In Wisden system [13], the author proposed a lightweight time-stamping approach in that it focuses on time-stamping the data consistently at the base station, rather than synchronizing clocks network-wide. This approach requires an addition small number of bytes to each packet instead of incurring additional messaging costs. In Wisden, each node calculates the amount of time spent by a sample at that particular node using its local clock. This amount is added to a residence time field (the additional bytes) in a packet as the packet leaves the node. The delay from the time of generation of the sample to the time it is received by the base station is stored in the packet as the sample travels through the network. The base station, equipped with a GPS receiver, can thus calculate the time of generation of the sample by subtracting the residence time from its local time. However, the clock drift will affect the precision of the time stamp. The longer the packets stay in the network,
the more the drift affection is. In both FTSP and Wisden, the propagation time is neglected.

3 Smart Sensing Component Design

3.1 Hardware Design

Currently, three types of sensor nodes are widely used in WSN systems. Among these sensor nodes, Mica family motes have 4K RAM space and 7.37MHZ CPU; MSP430 family motes have 8K RAM space and 8MHZ CPU; Both of these two family of motes need to work with EEPROM or Flash storage component to satisfy the high data rate demand. However, the operation time and energy consuming on the external storage medium is a constraint for real-time applications. In order to meet the needs of timeliness and high data rate, we chose the Imote2 sensor node for the OASIS prototype box. The Imote2 sensor node has attractive features that offer better in-network processing capability: the CPU core frequency of Imote2 is feasible to be selected from 13MHz to 416MHz; Imote2 also provides 256K SRAM and 32M SDRAM memory space; three SPI interfaces, three UART interfaces, and multiple GPIO interfaces available on Imote2 enable flexible extension. In order to gain high data resolution, a 16-bit or 24-bit AD chip is always used in environment monitoring systems. Thus we chose the MDA320CA sensor board as the sampling processing board for our prototype box. MDA320CA sensor board is equipped with Texas Instrument ADS8344 analog-to-digital conversion chip. The data resolution of ADS8344 is 16-bit, and it supports eight single-ended or four double-ended analog input channels. The MDA320CA sensor board also provides eight-channel digital I/O for digital sensor connection.

Though the Global Positioning System (GPS) receiver is an energy consuming component, it is still necessary for the application like Zebranet [8], which requires the node location information. Besides tracking sensor nodes, the GPS receiver is often utilized for time synchronization purposes. Normally, the sink node in a wireless sensor network is equipped with a GPS receiver to be the time root, and other nodes synchronize with the sink by running a time synchronization protocol. In the OASIS project, a GPS receiver is used for both time synchronization and node localization purpose. The GPS receiver in our prototype box is a low power consuming GPS produced by U-Blox.

We also designed the interconnection board to connect sensor board, sensor node, GPS receiver and other external components. The interconnection board also extends the existing interfaces on Imote2 and MDA320CA for future utilizing. For example, we extended the default UART interface of Imote2 to communicate with PC, and a transparent radio is connected to this UART interface in the sink node. Thus this antenna works as a virtual PC and it could utilize the existing Imote2-to-PC communication software without any change.

The hardware component connection relationships are shown in Figure 1. Imote2 connects with MDA320CA sensor board through SPI2 interface and the ADS8344 SPI clock signal is controlled by Imote2. The GPS receiver is connected to Imote2 through Blue Tooth UART interface, and GPIO93 on Imote2 is reserved for pulse-per-second (PPS) signal capturing. In order to increase the radio range, we connected an external antenna to the the CC2420 radio chip on Imote2. The GPS receiver, Imote2 sensor node, MDA320 sensor board, and the interconnection board are placed inside a weatherproof iron box, see Figure 2. Different components must be strictly ground-power isolated and the electromagnetic interference problem must be avoided.

3.2 Configurable Sensing

Highly configurable sensing is a fundamental service in our smart sensing design. This configurable feature includes both self-adaptive configuration and user interactive configuration. The user configuration is implemented by RPC command&control developed in our previous work [15]. The configuration operation may happen at system setup, detection of system alarm and network congestion. The configurable services will cover following items:

3.2.1 Sensing Data Management

The sensing data management module is in charge of saving sensing data into different storage mediums based on platform attributes and network situations. If there is enough RAM space free to use, sensing data will be saved into RAM space instead of Flash chip. However, network con-
gestion or other problems may cause a tight memory resource condition. After detecting these situations, the data management module needs to move unsent data left in RAM into external storage medium for later retrieval. For platforms with less RAM space like the Micaz mote, the system needs to save data into an external storage component more frequently to avoid data loss risk. For platforms with large RAM space like Imote2, it is efficient to allocate a big buffer space in RAM for sensing data management and move old data into Flash only if the memory usage level is over warning threshold. In general, the main function of the generic data management module is to allocate free buffers for sensing data and save sampled data into a safe place for later retrieval. The data management module needs to hide the implementation details of how a free buffer is allocated to the sensing module and where the sensing data is saved.

In our implementation, a list of sensor blocks are allocated initially (To follow TinyOS convention, we allocate fixed buffer space in RAM with system starting up. The buffer could be dynamically allocated if the operating system supports it). Each of the sensor blocks, see Figure 3, has the information of sampling start time, sampling rate, task execution code, status and a raw buffer to save sensing data. Only when it has been requested by the sensing module for a free buffer, one unused block is allocated and the block status will change from FREE to FILLING. After saving MAX_BUFFER_SIZE sample data, the status will change to a pending status which is recognized by different processing tasks (will be discussed in section3.2.3). The sensor block will be freed after it has been processed. There is a corresponding queue to manage blocks in the same status and the next and prev field is used to connect these blocks. If the block list is three-quarters full, the first quarter of unsent data will be moved to SDRAM.

### 3.2.2 Online-Configurable Sensor Driver

The sensing module in Figure 4 maintains information of different sensors including sampling rate, sensing channel which could either be ADC channel or digital I/O, sensing data resolution, sensor status, reference voltage gain. All of these parameters could be tuned according to system status to improve system performance and data validity. For example, seismic sensor priority could be increased to allocate more bandwidth for seismic data than other types of sensors when an earthquake event is detected. Even though all the nodes in the network are running the same application software, they could dynamically add or delete some type of sensor without reprogramming the whole network. For example, all the nodes are initialized by turning on two types of sensors: seismic sensor and infrasonic sensor. Users might add a new sensor, for example, a SO2 sensor, to the node later. It is not necessary to reprogram the software to accommodate to such a change, but just plug in the sensor and send one configuration command to start the corresponding sampling task. Also, it is possible to remove a sensor after faulty detection. As discussed in section 3.2.1, our design requires no fixed memory allocation for each sensor to maximize the memory usage.

### 3.2.3 Configurable data processing schedule

After system starts up, users can change the execution sequence of different sensing data processing tasks. For instance, two event identification tasks have been programmed into the node; the default one is short-term/long-term (STA/LTA) average trigger algorithm and the other unused one is level-trigger event detection algorithm. Users can select which algorithm to be run on the sampled data dynamically. Not only selection, but also other scheduling operations like deletion, adding and reordering are supported. Assuming that eight different tasks with ID 1 to 8 have been programmed into the node and the default data processing task list is (1, 2, 3, 4). This list could be con-
figured to (3, 4, 6, 2, 1), (1, 6, 2) or any other permutation, see Figure 5. In section 3.2.1, we have mentioned that sensing data is managed by sensor block structure. Each of the blocks will be assigned a default execution code when allocated (Figure 3), and each task can only get the data blocks in the corresponding status. For example, the compression task could only get data from the queue managed blocks in status TOBECOMPRESSED and the event detection task could only get data from the TOBEDETECTED queue. If TinyOS supports dynamic module insertion as Linux, a new task binary image could be sent to the node to extend the existing processing tasks.

3.3 Signal Processing

The processing module runs different processing tasks on sensing data managed by the data storage module. The purpose of the processing module is to optimize network reconstruction under different system situations. The processing tasks include error detection, event detection, prioritization or other smart sensing relevant tasks. This module could configure other modules adaptively based on the analyzing result. In order to implement correlated event detection, fault tolerance and data aggregation, the processing module could also overhear neighboring data messages.

3.3.1 Fault Detection

Our implementation for fault detection is based on statistic information (maximum, minimum, average, deviation) of block-wide data sets. Several different error models usually exist in sensing components are listed here.

- Sensor board disconnection. The maximum, minimum and average value will be the same, 0xFFFF. This error could happen due to unstable sensor board connection.

- Broken Sensor. Broken sensor error could be caused by sensor disconnection in which case the ADC channel might sample random data similar to amplified noise; Sensor damage that might present a symptom of the environment change on the data is also broken sensor error. A complex pattern recognition algorithm or collaborative error detection mechanism as mentioned in section 2 could be applied to identify the broken sensor.

- System alarm detection. System alarm includes low battery voltage, low free memory space and other serious situation for sensing component. A batch timer could be started to check battery, memory, radio or other component to update alarm status.

Faulty sensor detection is not trivial to be implemented based on single node information due to the environment affections. The predefined error pattern could be mixed up with the background noise and lead to false detection. In our current implementation we have developed the diagnose rules for system alarms: 1) battery voltage is less than 3.5 volts. 2) sensor block list is threequarter full 3) radio is busy for more than ten seconds. 4) GPS data is invalid for 120 seconds. The system will also check the board disconnection error. The broken sensor error identification will be addressed in our next step of implementation.

3.3.2 Denoising

When the S/N ratio is low, the event detection and fault detection algorithm may report false analyzing results, thus the denoising processing is necessary. In addition, denoising will produce more similar data set and it is useful for data compression processing. Denoising could utilize hardware filters, software filters or thresholding techniques. In our system, we implemented denoising by soft thresholding since it has a better mathematical property than hard thresholding [3] (t denotes the threshold. The soft threshold signal is \( \text{sign}(x)(|x| - t) \) when \(|x| > t\) and 0 when \(|x| < t\). The hard threshold signal is \( x \) when \(|x| > t\) and 0 when \(|x| < t\).

3.3.3 Data Prioritization

Distributed event detection is an important feature in smart sensing; it could enable efficient node resource utilization and high fidelity network reconstruction in the base station. Different event detection algorithms have been developed. The simple detecting method like level trigger simply compares the amplitude of each sample to a preset threshold. Event recording starts whenever the threshold is reached and stopped when the level is below the threshold. The simple integer comparison makes it fast and efficient. However, a learning of stream sampling data is necessary to detect complex events. In our current implementation, we use the short-term average long-term average (STA/LTA) trigger algorithm to locate the earthquake event period in each sensor nodes. STA/LTA is a general event detection algorithm in seismic data analyzing.

In our implementation, continuous sensor blocks with seismic data are processed by following equations:

\[
X_i = \sum_{j=0}^{n-1} x_{i-j}
\]

\[
x_i = |d_i - d_{i-1}|
\]

\[
X_i = X_{i-1} + \frac{x_i - x_{i-n}}{n}
\]

where \(d_i\) is the sensing value in the block; \(n\) is the number of points in the STA or LTA windows; \(X_i\) is the average value; STA and LTA value are decided by different \(n\). The absolute average STA over the STA time window is determined and the same signal is used to calculate the LTA over the LTA time window. Thus, LTA will give the long term background signal level while the STA will respond to short term signal variations. The ratio between STA and LTA is constantly monitored and the start of an event is declared once it exceeds the trigger level. The LTA is frozen after the event starts, so that the reference level is not affected by the event signal. The end of the event is declared when the STA/LTA ratio reaches the de-trigger level. The values of
trigger level and de-trigger level is 4.0 and 2.0 respectively. However, the STA/LTA trigger usually does not function well with high and irregular seismic noise.

Another technique, applying AIC autopicker on wavelet transform of seismic data [16], has been developed to identify the event arriving time. However, the computation complexity make it unfeasible to do distributed processing in wireless sensor network systems.

In our system, we assigned different priorities to different types of data based on the event detection result, and the lower layer module will allocate transmission opportunities based on the priority. If an earthquake event is identified, the data set within the event trigger period will be marked a high priority. The error report and self-configuration report are also high priority messages.

3.4 Time Synchronization

Before describing the robust time synchronization design, we need to introduce the real time clock (RTC) module in the OASIS system (Figure 4). Firstly, the RTC module is driven by a finer grain clock which fires every 1000 microseconds. The RTC module maintains a time count as local time which is increased by one when the micro clock fires. Secondly, the RTC module provides a generic timer interface which supports one minimal millisecond fire interval. The sensing module will use this interface to control sampling but all the sensors will only share one timer channel. The timer fire interval of the sensing module is the greatest common denominator of all sensors’ sampling rates. This design solves the problem of ADC channel races and uses less timer resources. Another attribute of the RTC module is that it is connected to time-synchronization module for synchronized sampling and time-stamping services. The robust time synchronization design is composed of three parts: time synchronization, time stamping and the synchronized sampling.

3.4.1 Time Synchronization

Time synchronization could either be implemented on a time synchronization protocol such as FTSP, or GPS signal processing. In our system, the hybrid mode time synchronization is developed to guarantee robustness and faunal is captured after preparing the UTC time, the time count in RTC module is immediately set with previous ready-to-use UTC time. By doing this, the delay of receiving GPS data and extracting time information can be avoided.

However, the GPS receiver could be damaged as discussed in section 1. If a node loses valid GPS signal for a threshold time, that node will switch to FTSP mode automatically. For any node that is running in FTSP mode, it will send one reference-time requested message to its neighboring nodes. The neighboring nodes will broadcast time messages periodically following the FTSP mechanism after receiving the request. The FTSP messages will only be processed by those nodes running in FTSP mode and only GPS mode node can claim itself to be the time root. If all the nodes inside the network switch to FTSP mode, the user in the control center may decide which node to be the time root.

3.4.2 Time Stamping

The time stamp for each sensor block (Figure 3) is set at the first sampling of that block. The time information inside the sensor block structure is the time count value, and this count is translated into time stamp format, minute : second : millisecond : interval, only at the time of transmission.

3.4.3 Synchronized Sampling

We implemented the synchronized sampling by taking advantage of the real time clock module. Assuming all the nodes in the network are successfully synchronized, then all nodes could sample at the same UTC time to achieve synchronized sampling. The implementation has two steps: 1) the sensing module call timer interface of real time clock module to start sensing, and the fire interval is δ. 2) if current time is Tcurrent, the fire point Tfire is set to (Tcurrent + δ) instead of Tcurrent + δ. For example, δ = 10ms, Tcurrent is 20:00:00:422, then the next fire interval is 20:00:00:430 but not 20:00:00:432. After set Tfire for the first time, the successive Tfire is updated by adding δ to last Tfire for continuous sampling.

4 Conclusion

In this paper, we have proposed a smart sensing design solution for a volcano monitoring WSN system. This smart sensing component is helpful in improving resource utilization, situation awareness and system performance. Our solution has been implemented on the OASIS prototype box running TinyOS. The sensor block list implementation enables efficient memory space utilization and the sensing data management module hides the operation details among different storage mediums from a user. The configurable sensing attribute facilitates sensing component management and increases the system flexibility. Our time synchronization design satisfies the requirement of network-wide synchronized sampling with a high global time accuracy. The denoising processing and event-trigger mechanism helps to locate the important segment from large
stream sensing data and the priority mark on different types of data improves the network bandwidth usage, which enables the faithful network reconstruction in the end user side. Currently, we are using the DA convertor board (PCI-DAC6703) to evaluate the system performance. The seismic data file is read and the converted analog outputs work as real sensor inputs to sensor nodes. Our next research focus is to develop lightweight compression algorithm to compress the insignificant data transmission to reduce the bandwidth competition caused by a high data rate.

Acknowledgements

We greatly appreciate the support from NASA ESTO AIST program under the grant #NNX06AE42G.

References