SMART SENSING DESIGN FOR ENVIRONMENT MONITORING SENSOR NETWORKS

By

YANG PENG

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY VANCOUVER
School of Engineering and Computer Science

AUGUST 2008

To the Faculty of Washington State University Vancouver:

The members of the Committee appointed to examine the thesis of YANG PENG find it satisfactory and recommend that it be accepted.

Chair

ACKNOWLEDGEMENT

I would like to express my gratitude to my advisor, Professor WenZhan Song, for his guidance, inspiration and supervision of this thesis.

I would also like to take this opportunity to thank Rick LaHusen for his help and suggestions on the hardware and the situation awareness component design.

Many thanks to all the members in our team for their suggestions to improve this thesis and help on component evaluations. Without their generous help, continuous encouragement and moral support, this work could not have been completed.

YANG PENG

SMART SENSING DESIGN FOR ENVIRONMENT MONITORING SENSOR NETWORKS

Abstract

by Yang Peng, M.S.
Washington State University Vancouver
August 2008

Chair: WenZhan Song

An environment monitoring sensor network typically embeds in an unattended environment. Users demand long-lifetime healthy systems to continuously run in harsh environments and want to change the system configurations on the fly. The robustness, configurability, fault detection and recovery and situation awareness have become key system design challenges, because of the frequent damages to less-protected sensors and the demand of real-time high-fidelity data. In this thesis, a smart and robust sensing component design with configurable sensing, situation awareness and robust time synchronization components will be presented. The smart sensing component have been implemented and tested in an indoor testbed of sixteen prototype sensor nodes for volcano monitoring. The proposed architecture and framework can be applied to other wireless sensor network applications.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS iii

ABSTRACT iv

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

1. INTRODUCTION 1

1.1 Motivation 3

1.1.1 Necessity of Smart Sensing Component 3

1.1.2 Challenges for Smart Sensing in WSNs 3

1.2 Research Background 4

1.3 Goal 6

1.4 Thesis Outline 7

2. SMART SENSING DESIGN 9

2.1 Design Goals 9

2.1.1 Lightweight 9

2.1.2 Generic and Configurable 9

2.1.3 Intelligent 10

2.2 Architecture 11

2.2.1 Smart Sensing Component in OASIS 11

2.2.2 Smart Sensing Architecture 12

2.3 Smart Sensing Testbed 14

2.3.1 Lab Environment Mini-Net 14

2.3.2 GPS Signal Distribution 14

2.3.3 DAC Board 15

3. HARDWARE DESIGN 16

3.1 Seismic Sensor 16

3.2 Infrasonic Sensor 17

3.3 Lightning Sensor 18

3.4 GPS Receiver 18

3.5 iMote2 Platform 19

3.6 Sensor Board 20

3.7 Component Connection 22

4. CONFIGURABLE SENSING 23

4.1 Reliable Command Control 23

4.1.1 Reliable Broadcast Related Works 23

4.1.2 Cascades Protocol 25

4.1.3 Cascades Evaluation 32

4.2 Configurable Sensing Services 35

4.2.1 Parameter Tuning 35

4.2.2 Memory Management 36

4.2.3 Read/Write Parameters into Flash 38

4.2.4 Generic Support to Different Platforms 38

4.2.5 Configurable Sensor Selection 39

4.2.6 Configurable Processing Task Schedule 39

5. SITUATION AWARENESS 42
5.1 Event Detection Related Works 42
5.2 Environment Event Detection 44
5.2.1 Threshold Event Detection 45
5.2.2 STA/LTA Event Detection 45
5.2.3 Data Prioritization 47
5.3 Event Detection Evaluation 48
5.4 Fault Detection and Recovery 50

6. SYNCHRONIZED SAMPLING 52
6.1 Time Synchronization Related Works 52
6.2 Synchronized Sampling Implementation 54
6.2.1 Real Time Clock Module 54
6.2.2 Microsecond Resolution Clock 56
6.3 Time Synchronization 56
6.3.1 GPS Data Format 57
6.3.2 GPS Time Based Synchronization 58
6.3.3 FTSP Based Synchronization 59
6.3.4 GPS, FTSP auto-switch 60
6.4 Time Synchronization Evaluation 61

7. CONCLUSIONS 63
7.1 Main Contributions 63
7.2 Future Work 64

BIBLIOGRAPHY 65

LIST OF TABLES

Page

1.1 Sensor Type and Sampling Rate 5

3.1 Accelerometer Chip Specification 17

3.2 Infrasonic Sensor Specification 18

5.1 Data Type and Priority 48

5.2 Priority 50

6.1 UBX RXM-RAW Message Format 57

6.2 Hop Count Time Sync Error 62

LIST OF FIGURES

Page

1.1 In-Situ Network in OASIS Project 6

2.1 Smart Sensing in OASIS System 11

2.2 Smart Sensing Component Framework 12

2.3 Lab Environment Testbed 14

2.4 DAC Board 15

3.1 Sensor Chips 17

3.2 GPS Receiver 19

3.3 iMote2 Sensor Node 20

3.4 MDA320 Sensor Board 21

3.5 Prototype Sensor Box 22

4.1 Opportunistic Broadcast Flow with Cascades Protocol. 28

4.2 Possible Inconsistent Parent-Child Relationship during Route Updates 29

4.3 Cascades implementation in TinyOS 31

4.4 Reliability in Different Checking Time 33

4.5 Communication Cost in Different Checking Time 34

4.6 Longest Time Delay for Each Packet 35

4.7 Memory Block Data Structure 36

4.8 Processing Task Scheduling 40

5.1 Event Detection Result 49

6.1 Synchronization Process 54

6.2 NMEA Message Format 57

6.3 GPS Time Synchronization 58

6.4 Time Synchronization Error 61

CHAPTER ONE

INTRODUCTION

The Wireless Sensor Network (WSN) is a relatively new application of ad-hoc networks. WSNs are able to interact with environments by sensing or controlling physical parameters with inexpensive sensor nodes [24]. The WSN technology involves deploying a large number of sensor nodes over the area of interest. Each node is equipped with one or several application specified sensors, a short range radio and a low power consuming microprocessor (some may own external storage components for data logging). Sensor nodes perform self-organizing and self-healing to form and maintain an ad-hoc network, which is capable of transmitting the sensed data to one ore more base stations.

WSNs are powerful in that they are amenable to support lots of different real-world applications: the system for detection of specific events [25]; the intelligent building monitoring WSN system [19] to increase the comfort level of inhabitants and reduce the energy consumption. WSNs are also used in environment monitoring applications such as forest fire monitoring [18] and habitat exploration of animals [33]. Body WSN [30] is a very beneficial application by providing emergency health care alarm and long term patient and doctor tracking. WSNs can also be applied to military usages including battlefield surveillance and monitoring and detection of attack by chemical, biological, or nuclear weapons. Wireless Sensor Networks also have the potential to benefit our society in numerous ways [15]. With the increasing development of electronic components and advances in communication technology, WSNs have an enormous economical potential in future.

While providing high quality monitoring for large geographical areas, WSNs also post many research challenges due to following reasons [48]:

- In WSNs, all routing and maintenance algorithms need to be distributed to accommodate dynamic node deployments. A WSN should also be capable of adapting to changing connectivity due to the failure or new join of sensor nodes.

- Energy efficiency is a primary requirement in a WSN system design and all system components need to consider the available energy at the nodes as a major constraint.

- Real-time communication over sensor networks must be supported through provision of guarantees on maximum delay, minimum bandwidth, or other QoS parameters.

- Sensor nodes are randomly deployed and do not fit into any regular topology. Once deployed, they usually do not require any human intervention. Hence, the setup and maintenance of the network should be entirely autonomous.

In order to design an intelligent wireless sensor network, both hardware and software aspects need to be carefully considered. Widely deployed sensor node platforms include the MICAz mote which has 4K RAM space and 8K ROM space [3], the Tmote Sky sensor node [39] with 8K RAM and 40K ROM, and the new generation platform, Intel iMote2 sensor node. All of these nodes adopt Chipcon CC2420 [22] radio chip as the radio component, which is a 2.4 GHz IEEE $802.15.4$ and Zigbee compliant RF transceiver. In order to minimize code size as required by the memory constraints inherent in sensor networks and guarantee the real-time characteristic, several event driven and lightweight operating systems are developed for WSNs. TinyOS [27] is an open-source embedded operating system designed for WSNs featured a component-based architecture and enables rapid innovation and implementation. TinyOS is written in the nesC programming language (a dialect of the C programming language optimized for the memory limitations of sensor networks). Contiki [13] is another widely used OS for WSNs, and it has been ported to various hardware platforms and implements a TCP/IP stack on top of a platform with severely restricted resources.

1.1 Motivation

1.1.1 Necessity of Smart Sensing Component

In the past several years, the communication protocols of wireless sensor networks, such as routing [10,12], medium access control [9,43,59] and congestion control [44,53], have received much attention. Relatively, less attention has been paid to the sensing and middleware components. A sensor node, unlike a PC or PDA, usually works in the unattended environments and it might not receive regular maintenances from users. Hence the characteristics of robustness, situation awareness, fault detection and recovery are very important in deciding system performances [42]. For instance, if a broken sensor is not detected and suppressed, the node may transmit meaningless data and occupy the radio spectrum resources. The sensing component shall be intelligent to detect such a situation and notify the system to handle malfunctioning nodes and prevent potential system failures. In the volcano monitoring sensor-web project we are working on, various geophysical and geochemical sensors are equipped to generate continuous data representing different aspects of volcano activities, and data priorities depend all on the volcano status. An erupting volcano, like Mount St. Helens, provides a challenging environment to examine and advance the sensor network technology. The crater at Mount St. Helens is a dynamic 3-dimensional communication environment with batteries as the only reliable energy source and sensors are occasionally destroyed by eruptions. Additionally, in many earth science applications, scientists demand data with precise time-stamp to enable seismological analysis. There is also a compelling need for real-time data to enable emergent alarm generations. Hence, in order to collect high fidelity data for the scientific monitoring and analysis, a smart sensing component is necessary and important.

1.1.2 Challenges for Smart Sensing in WSNs

The sensing module in an environment monitoring system faces many challenges that is rarely been considered in other Wireless Sensor Network systems:

- High frequency, high resolution and real-time data collection: In contrast to sensor networks targeting low-rate data collection, 100 Hz frequency and 16-bit data resolution of multiple channel samplings (> 3) bring high workload of data processing and competition for processor resources to the system. Hence, tasks of sensor sampling and data processing (analyzing, logging and packaging) must be finished in a short period to guarantee the real-time characteristic. Sensor sampling rates in the evaluation system are showed in Table 1.1, and the raw data rate reaches 400 Bps, without including message headers and other control messages.

- Synchronized sampling: In order to compare monitoring data across the network, seismologists hope to see all nodes in the network sample the same sensor channel synchronically, and the sensing time differences must be less than one millisecond. Though a node could be equipped with a GPS receiver to achieve a high accurate time-stamp, the network-wide synchronized sampling is still not absolutely guaranteed due to multiple time delay sources and occasional GPS signal failures.

- Event detection accuracy: In order to reduce network traffics, a system could stop real-time data transmission for a short period; however, the system still needs to detect all big events and send corresponding event data back to the control center during this period. Distributed event detection algorithms need to be developed for accurate events triggering.

- Harsh environment: The changing environment like an active volcano is another big challenge for the sensing component. Sensors can more easily be destroyed than in general situations and valueless data from broken sensor nodes will affect the whole system. It is necessary to implement error detection and recovery scheme to reduce the error harm.

1.2 Research Background

The research topic of this thesis comes from the Optimized Autonomous Space-In-situ Sensor-web (OASIS) project [5]. The goal of this project is to develop a prototype dynamic and scalable hazard monitoring *Sensor-web* [11].

Sensor Data Type	Sampling Rate (Hz)	Resolution (Bytes)	Bandwidth (Bps)
Seismic Sensor	100	2	200
Seismic RSAM [40]	1	2	2
Infrasonic Sensor	100	2	200
Infrasonic RSAM	1	2	2
Lightning Sensor	1	2	2
GPS	0.1	$200 \sim 300$	20

Table 1.1: Sensor Type and Sampling Rate

A Sensor-web is a coordinated observation infrastructure composed of a distributed collection of resources (e.g., sensors, platforms, communication infrastructures) that can collectively behave as a single, autonomous, dynamically adaptive and reconfigurable observing system that provides raw and processed data, along with associated meta-data, via a set of standards-based service-oriented interfaces [46].

Sensor-web related research spans multiple domains, including distributed systems, wireless sensor networks, remote sensing, artificial intelligence, sensor-web services and etc. A multidisciplinary team involving computer scientists, space scientists, and earth scientists will work on this project. The final self-configuring self-healing remote Sensor-web will be applied to one of the most active volcanos - Mount St. Helens. The Figure 1.1 shows the in-situ network structure after the deployment of sensor nodes.

The two non-exclusive reasons of scientists monitor volcanoes includes: (1) to monitor hazards by assessing the level of volcanic unrest; and (2) to understand physical processes occurring within the volcano, such as magma migration and eruption mechanisms [36, 45]. The type of instrumentation used to study volcanoes depends on the science goals of the deployment and we are focused on the use of wireless sensor network for long-term field deployments involving dozens of sensor stations deployed around an expected earthquake source region.

Benefits of wireless sensor networks have been recognized by more and more earth scientists. In testimony to Congress in May 2005, NASA included the following statement: "In the future,

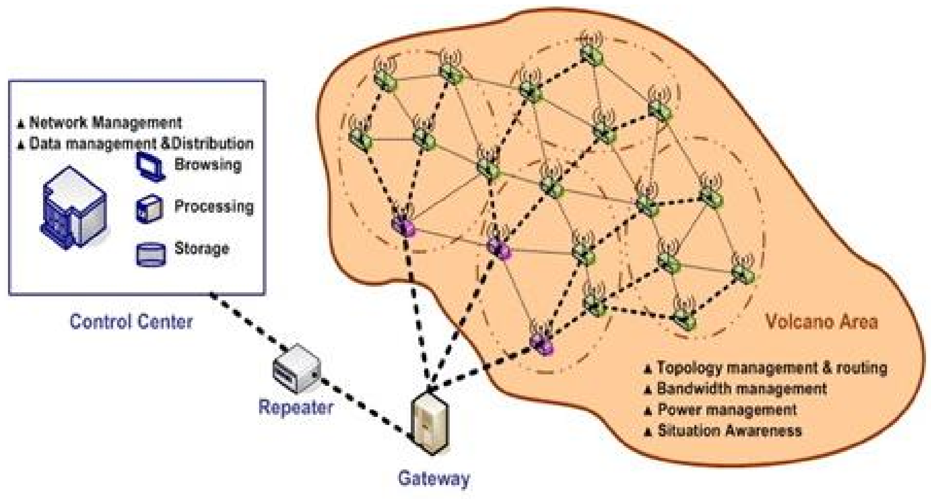


Figure 1.1: In-Situ Network in OASIS Project

NASA plans to develop a sensor web to provide timely, on-demand data and analysis to users who can enable practical benefits for scientific research, national policymaking, economic growth, natural hazard mitigation, and the exploration of other planets in this solar system and beyond." NASA Earth Science Technology Office (ESTO) studies addressing sensor-web concepts can be found at the ESTO website [4].

1.3 Goal

The primary goals of this Optimized Autonomous Space In-situ Sensor-web (OASIS) are: 1) to integrate complementary space and in-situ (ground-based) elements into an interactive, autonomous sensor-web; 2) to advance sensor-web power and communication resource management technology; and 3) to enable scalability for seamless infusion of future space and in-situ assets into the sensor-web.

Considering the sensing component design for volcano monitoring, sensing processing packages are designed to be placed in weatherproof enclosures for the harsh environments. In software, a scalable memory management mechanism and configurable sensor driver are designed to allow users to configure the sampling rate, ADC channel, and specify the data processing tasks on different type of sensors at anytime after the deployment. A situation awareness middleware is also developed based on mature volcano signal processing algorithms to detect volcanic events and adjust packet priorities to ensure that more important data reach users earlier with better opportunities. To enable accurate time-stamping on sensor data, the robust time synchronization mechanism is also developed.

The general description of Smart Sensing component design is given out here after analyzing challenges in Section 1.1.2, and design details will be presented in Section 2.

1. Time Synchronized Sensing: With the help of GPS receiver or time synchronization protocols, coordinating the network to perform sensing action synchronically and adding accurate time-stamp to sampling data.

2. Intelligent Signal Processing: Node level lightweight data processing to identify volcano events and help to reduce data transmissions by only sending out big events data while keeping silent for insignificant events; Compressing non-significant data when network traffic is congestive.

3. Interactive and Reliable Control: Reliable and fast delivery of control commands from uses to target nodes in the network.

1.4 Thesis Outline

Chapter 2 first describes the smart sensing design framework and introduces the role of smart sensing component in the OASIS system. This chapter ends by introducing the evaluation environment for smart sensing component.

Chapter 3 focuses on the hardware design of prototype sensor node boxes, which covers the design of three different sensors (seismic, infrasonic and lightning sensors), the GPS receiver, the sensor board and the component connection board.

Chapter 4 covers the configurable characteristic of smart sensing component and the reliable command and control service. At first, the reliable broadcast protocol - Cascades, which guarantees the reliable delivery of control command to individual nodes, is discussed. The second part of this chapter is about the configurable sensing feature which includes parameter tuning, memory management, configurable sensor driver and configurable data processing task schema.

Chapter 5 discusses the design and implementation of situation awareness idea in smart sensing component. Such situation awareness based signal processing includes fault detection, de-noising and event detection processing.

Chapter 6 describes how to perform network-wide synchronized sampling and add accurate time-stamp to sensing data. This chapter also addressed the auto-switch mechanism between GPS receiver based time synchronization and protocol based time synchronization.

Chapter 7 concludes the thesis. It discusses the advantages and shortcomings of current design and implementation, and it also notes several areas of future work.

CHAPTER TWO

SMART SENSING DESIGN

2.1 Design Goals

2.1.1 Lightweight

Lightweight is one of the most important characteristics in smart sensing design, which addresses the real-time property and memory space constraints.

The microprocessor on a sensor node is generally a low frequency processor that consumes less power. However, due to the non-preemptive task attribute in WSN operating systems like TinyOS, if there is a time consuming task such as a floating-point calculation loop seizing the CPU, other tasks would be held in the pending status for a long time. Such situations could decrease the system timeliness. Therefore, efficient processing algorithms and short task routines need to be developed.

Smart sensing design also needs to handle the constraint brought by small memory spaces. Additionally, most of the existing operating systems for wireless sensor networks are weak in supporting dynamic memory allocation. The requirement of static memory allocation in a global scope limits the flexibility of the system program design. An efficient memory usage mechanism is necessary to support both sensing data storage and localized data analysis.

2.1.2 Generic and Configurable

The smart sensing design shall be integrated into applications developed on different platforms as a generic sensing component but not specified for one type of sensor node, thus it needs to provide a service that hides low level details of different hardware platforms. For instance, the Tmote Sky sensor node has an internal Analog to Digital Convertor while the iMote2 sensor node misses it. The sensing component needs to hide hardware details of ADC chip (channel allocation, reference voltage selection, etc) but just provides a generic interface for users to acquire sensing data.

Besides the generic ability to be reusable in different hardware platforms, the adaptability to various WSN applications is also desired. While the focus of the smart sensing in the OASIS project is on geophysical monitoring at volcanoes, the technology developed can be applied to a broad range of scientific, industrial, and medical uses of sensor networks. In particular, the stringent science requirements for geophysical monitoring, including the use of high-rate sensor data, reliable command and control, and synchronized sampling, are shared across many domains, thus the smart sensing design is expected to find value in these applications' areas as well.

Due to the important role that each individual sensor node plays in the overall network operations, it is vital that all sensor nodes receive configuration command within a relatively short period of time. For instance, in a volcano monitoring application, science agent software might want to change all seismic sensors' sampling rate from 100 Hz to 200 Hz in some hot area. It is necessary that this command reaches all target stations in a short time period. Therefore, the reliable configurability attribute is important for users to control the sensing behavior. A reliable command and control mechanism shall be developed to guarantee the delivery of each command from control center to individual nodes in the network. It is also necessary to let sensor node itself conduct autonomous adjustments according to environment changes and varying network statuses.

2.1.3 Intelligent

The OASIS project provides an important opportunity for geophysicists and computer scientists to optimize the information that is produced by an intelligent network of sensor arrays. One of the challenges as mentioned in 1.1.2 is that the data from the distributed networks must be reduced at each sensor node, yielding high-level data products that embody an interpretation of the volcano's activity. This is a drastic difference in the current approach, which requires all time-series data to be telemetered for subsequent reduction by human analysts. Smart sensing components will enable orders-of-magnitude augmentation in the number of sensor nodes and the lifetime of the networks. With enhanced sensor node distribution, geophysicists will be able to accomplish many goals that are not currently possible with traditional deployments.

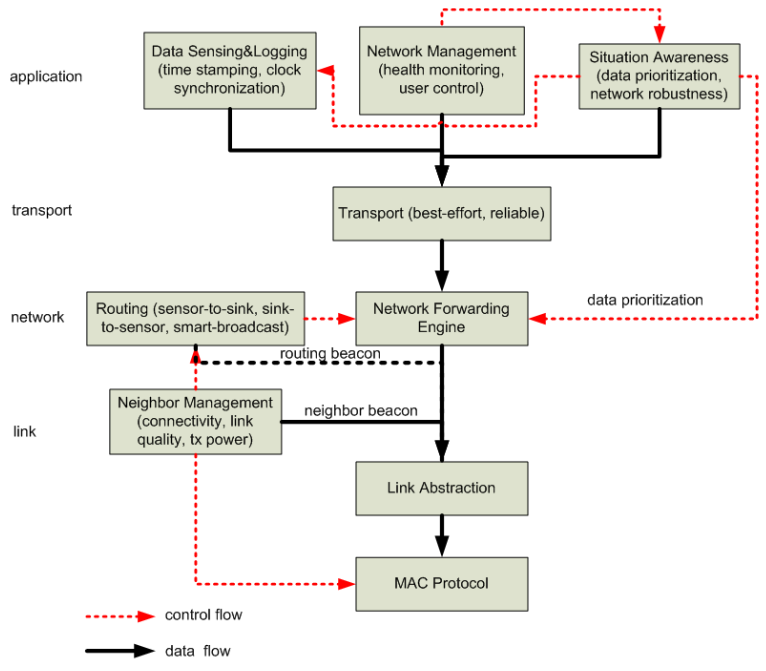

application
transport
network
link
Data Sensing&Logging (time stamping, clock synchronization)
Network Management (health monitoring, user control)
Situation Awareness (data prioritization, network robustness)
Transport (best-effort, reliable)
Routing (sensor-to-sink, sink-to-sensor, smart-broadcast)
Network Forwarding Engine
data prioritization
routing beacon
Neighbor Management (connectivity, link quality, tx power)
neighbor beacon
Link Abstraction
MAC Protocol
control flow
data flow

Figure 2.1: Smart Sensing in OASIS System

Node level data analysis includes series of signal processing techniques: 1) event detection, identifying user-concerned event data segments from sensing data stream; 2) prioritization, assigning different priorities to sensing data based on sensor types and weight levels (event, non-event, occasional, regular, etc); and 3) error detection and recovery, capturing abnormal sensing situations and removing valueless data from the source. The aim of such data processing series is to decrease the network bandwidth competition, thus reduce power consumption, expend system working period and increase data fidelity.

2.2 Architecture

2.2.1 Smart Sensing Component in OASIS

Before discussing the internal modules of smart sensing component, its role in the whole OASIS system is described here. According to the ISO network layer definition, the Smart Sensing component could be viewed as an application layer component. In Figure 2.1, three separate modules compose the whole application layer. The smart sensing component mainly covers the

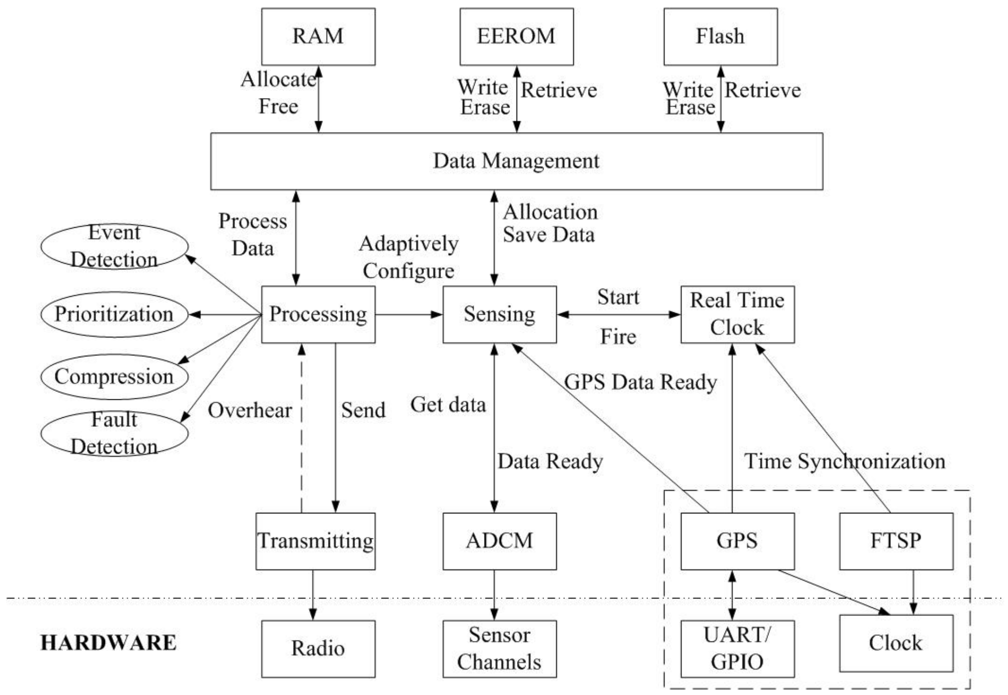


Figure 2.2: Smart Sensing Component Framework

Data Sensing/Logging module, and it interfaces with Network Management module through the RPC configuration mechanism. It also includes some part of the situation awareness features as described in Section 2.1.3.

2.2.2 Smart Sensing Architecture

The smart sensing component contains several key modules: Sensing, Real Time Clock (RTC), Time Synchronization, Data Management and Processing. Figure 2.2 depicts its internal framework.

Data Management module is in charge of monitoring and maintaining memory space usages on a sensor node. The details of the specific type of memory is hidden from sensing module. When the sensing module issues a memory allocation request, the Data Management module needs to return a memory block (see Section 4.2.2) in a ring buffer, which is allocated in RAM space. And the sensed data are finally stored into different mediums including RAM, EEPROM and FLASH space depending on network situations and hardware characteristics.

Sensing module focuses on acquiring sensor data from multiple ADC channels and maintains all the information about sensor statuses and sensing parameters. The sensing action is driven by timer events signaled by Real Time Clock module. Sensing module is also connected to the processing module, and the functionality is, instead of a direct data transferring, adaptively adjusting sensing parameters configuration decided by the Processing module based on the situation awareness analysis result.

Processing module implements a part of situation awareness functions by running a batch task to apply different processing algorithms on memory blocks in different statuses: `FREE`, `FILLING`, `FILLED`, `PROCESSING`, `PRESENT`. Status `FREE` means the memory block is not used now, status `FILLING` means sensing data is being filled into the current block, a block in status `FILLED` has been filled full, status `PROCESSING` means the block is under processing, and `PRESENT` is the last status indicating that the block is allowed to be sent out. After been processed in the Processing module, processed data will be packetized into active messages and relay to next hop node in the network. If the Processing component is connected to the receiver interface of the Transmission component, it is able to overhear neighborhood data for future correlation information based data processing.

Real Time Clock module provides a millisecond resolution timer to sensing module or users of other components (e.g., used by TDMA component as the time event generator). The timer service provided by this RTC module is different from default timer services in TinyOS or other WSN operating systems (see Section 6.3) and it satisfies the synchronized sampling demand. The RTC module maintains a 32-bit time count which is increased every 1000 microseconds driven by low level microsecond resolution clock, and this time count is the accurate global time after been initialized by time synchronization module. The local time count value is used for time-stamping.

Time Synchronization module could be implemented by a single protocol or a combination of several synchronization methods. If a node is equipped with a GPS receiver, this module could be controlled only by GPS and no other time synchronization protocol is needed. However, as GPS

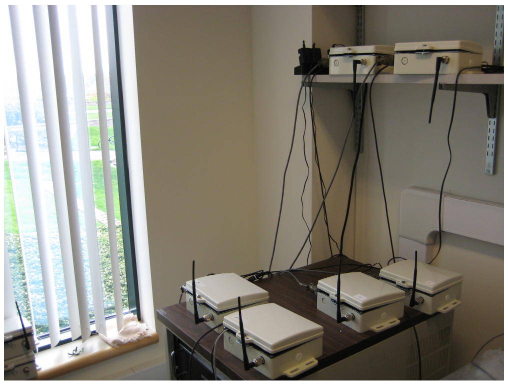


Figure 2.3: Lab Environment Testbed

is energy consuming and it is not guaranteed to work robustly (e.g., hardware broken), hence a hybrid mode of time synchronization mechanism is implemented in the OASIS system.

2.3 Smart Sensing Testbed

2.3.1 Lab Environment Mini-Net

In order to test the sensing component, a mini-net testbed with ten prototype sensor node boxes was build in SensorWeb Research Laboratory. The radio power of each box is set to the lowest level to form a multi-hop indoor network (Figure 2.3).

2.3.2 GPS Signal Distribution

One GPS Signal Distribution Unit is used to split GPS signal received from an outdoor GPS receiver due to the loss of GPS signal inside buildings. This signal distributer could serve up to sixteen connectors at the same time, and each connector will have different GPS signals. Such distributed GPS signals simulate the situation on volcanoes that sensor nodes placed in different positions receive different GPS signals.

Besides GPS signal, each boxes also needs to be connected to sensors. In order to make quantitative analyzing on data processing algorithms, all boxes are feed with the same user known sensor input. The sensor node analog channels real environment monitoring data in which the waveforms, event counts have been defined and it is easy to evaluate processing algorithms running in

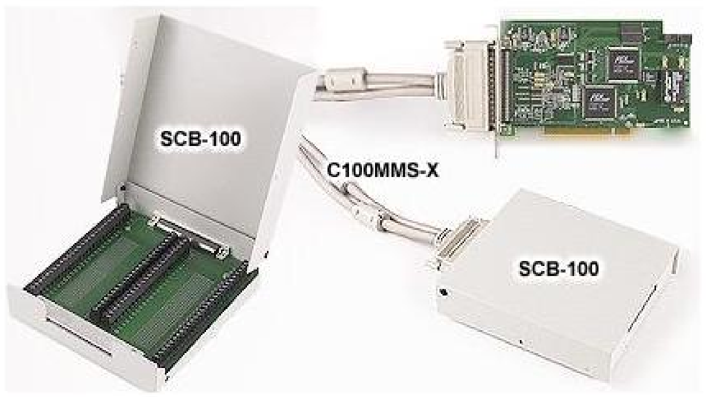


Figure 2.4: DAC Board

the network by analyzing received data in the control center.

2.3.3 DAC Board

Predefined signals used to feed ADC channels are generated by using PCI-DAC6703 board, which is a high-resolution analog output and digital I/O board designed for PCI bus-compatible computers. It provides 16 analog outputs and the voltage output range is ± 10.1 volts. The PCI-DAC6703 board has eight TTL-compatible digital I/O channels that are configured as one eight-bit port and each channel can be configured individually as either input or output. The PCI-DAC6703 board is also equipped with an onboard temperature sensor that measures the air temperature flowing over the board. Each board features plug-and-play installation and on-board digital calibration. All I/O signals are transmitted through a 100-pin connector.

CHAPTER THREE

HARDWARE DESIGN

Hardware design is a part of the smart sensing design to help increasing sensor node robustness, providing high fidelity sensing data and easing software processing complexity. The OASIS prototype box has following features aimed to satisfy the requirement of volcano monitoring:

- Fully self-contained package
- Analog, digital and serial interfaces for sensors
- MEMS seismometer for seismicity
- Infrasonic micro-barometer for explosion detection
- Lightning detector for severe ash cloud detection
- GPS receiver for deformation monitoring and precise system timing

3.1 Seismic Sensor

The most common instrument used in volcano monitoring is the seismometer, which measures ground propagating elastic radiation from both sources internal to the volcano and on the surface [36] (e.g., expansion of gases during an eruption).

The seismic sensor for the OASIS prototype box is a low noise MEMS accelerometer: Silicon Designs Model $1221J-002$ [47]. The Model 1221 is a low-cost, integrated accelerometer for use in zero to medium frequency instrumentation applications that require extremely low noise and the $2g$ version is ideally suited for seismic applications. Each sensor package combines a micro-machined capacitive sense element and a custom integrated circuit that includes a sense amplifier and differential output stage. It is relatively insensitive to temperature changes and gradients. The sensor specification is listed in Table 3.1.

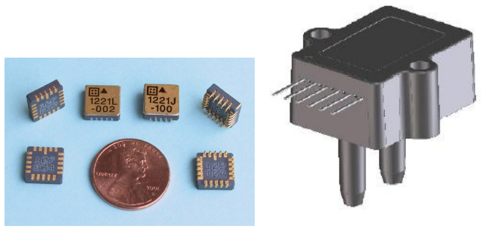


Figure 3.1: Sensor Chips

Specification Name	Value	Unit
Input Range	± 2	g
Frequency Response (Nominal, 3 db)	$0 \backsim 400$	Hz
Sensitivity (Differential)	2000	mV/g
Output Noise (Differential, RMS, typical)	5	μg/(root Hz)
Max. Mechanical Shock (0.1 ms)	2000	g
Working Temperature	$-55 \backsim +125$	C

Table 3.1: Accelerometer Chip Specification

3.2 Infrasonic Sensor

Infrasonic sensors are sometimes employed to record infrasound, low-frequency (< 20 Hz) acoustic waves generated during explosive events. Infrasound is useful for differentiating shallow and surface seismicity and for quantifying eruptive styles and intensity [23].

The infrasonic sensor - Model 1 INCH-D-MV, is a low range differential pressure sensor [6]. This millivolt output pressure sensor is based upon a proprietary technology to reduce all output offset or common mode errors. Output offset errors due to change in temperature, stability to warm-up, stability to long time period, and position sensitivity are all significantly reduced when compared to conventional compensation methods. In addition, the sensor utilizes silicon, MEMS, stress concentration enhanced structure to provide a very linear output to measured pressure. The sensor specification is listed in Table 3.2.

Specification Name	Value	Unit
Operating Range, differential pressure	1.0	H2O
Common-mode pressure	± 10	psig
Output Span, @ "H2O,	$9.0 \backsim 10.0$	mV
Offset Temperature Shift ($0 \backsim 50$ C)	± 250	μV
Offset Long Term Drift (one year)	± 250	μV
Working Temperature	$-25 \backsim +85$	C

Table 3.2: Infrasonic Sensor Specification

3.3 Lightning Sensor

Lightning occurred associated with the ash clouds of volcano eruptions [37]. Lightning was detected on seismograms as simultaneous spikes and simultaneous gain-ranging (a feature that normally lowers the gain at a station when the signal level begins to saturate) and the typical spike durations is $0.04 - 0.05$ s. Thus lightning strength correlates with both tremor amplitude and magmatic gas content. In general, volcanic lightning is important because it can help confirm that explosive eruptions are in progress, although the value of the information may be limited by the long delay from eruption onset to first lightning and the variability of eruptive and atmospheric conditions.

The OASIS lightning sensor is a simple RF pulse detector that triggers a one second digital pulse for easier acquisition and it shall be capable of detecting strikes from at least 10 Km from sensor.

3.4 GPS Receiver

USGS (U.S. Geological Survey) uses the GPS to measure the ground deformations in volcanoes. Thus the low power consumption U-Blox GPS receiver is chosen to be used in the prototype platform design. Naturally, the UTC time from GPS is used to timestamp sensing data.

The LEA-4T [51] as the core GPS signal processing chip of the U-Blox receiver features a Time Mode function whereby the GPS receiver assumes a stationary 3D position, whether programmed

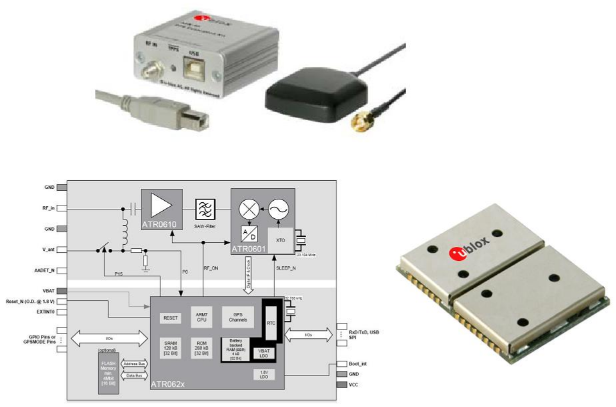


Figure 3.2: GPS Receiver

manually or determined by an initial self-survey. The accuracy of the time pulse is as good as 50 ns, synchronized to GPS or UTC time. An accuracy of 15 ns is achievable by using the quantization error information to compensate the granularity of the time pulse The LEA-4T also supports raw measurement data which is used in the OASIS system to offer precision positioning, real-time kinematics and attitude sensing.

3.5 iMote2 Platform

Currently, three types of sensor node are widely used in sensor network community. Among these sensor nodes, MICAz motes have 4K RAM space and 7.37 MHZ CPU; MSP430 family motes have 8K RAM space and 8 MHZ CPU. For those high-fidelity sensing application, due to limited RAM, these two families of motes need to buffer data to EEPROM or Flash storage component. However, the time delay and energy consumption on the external storage medium is a constraint for real-time applications. In order to meet the requirements of timeliness and high data rate, we chose the Intel Mote 2 (iMote2) to build the hardware prototype.

The iMote2 sensor node [1] is an advanced wireless sensor node platform. The platform is built around a low power XScale processor, PXA271. It integrates an 802.15.4 radio (ChipCon 2420) and a built in 2.4 GHz antenna. The iMote2 platform is a modular stackable platform and can be

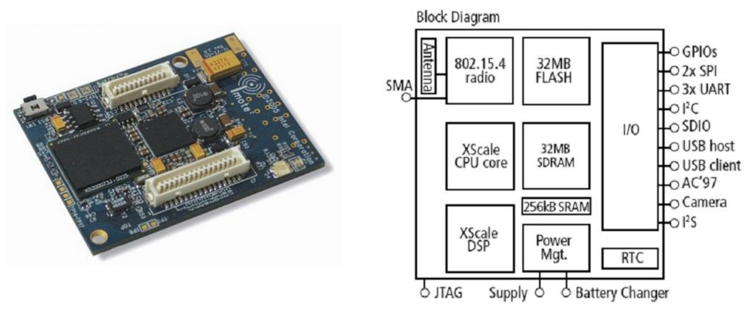


Figure 3.3: iMote2 Sensor Node

stacked with sensor boards to customize the system to a specific application, along with a "power board" to supply power to the system.

The PXA271 processor on iMote2 can operate in a low voltage (0.85 V) and a low frequency (13 MHz) mode, hence enabling low power operation. The frequency can be scaled to 104 MHz at the lowest voltage level, and can be increased up to 416 MHz with Dynamic Voltage Scaling. It also integrates 256 KB of SRAM divided into 4 equal banks of 64 KB. The PXA271 is a multi-chip module that includes three chips in a single package, the processor, 32 MB SDRAM and 32 MB of flash. The processor integrates many I/O options making it extremely flexible in supporting different sensors, A/Ds, radio options, etc. These I/O options include I2C, 3 Synchronous Serial Ports one of which dedicated to the radio, 3 high speed UARTs, GPIOs, SDIO, USB client and host, AC97 and I2S audio codec interfaces, fast infrared port, PWM, and Camera Interface. The processor also adds many timers and a real time clock. The PXA271 also includes a wireless MMX coprocessor to accelerate multimedia operations. It adds 30 new media processor instructions, support for alignment and video operations and compatibility with Intel MMX and SSE integer instructions.

3.6 Sensor Board

The MDA320CA sensor board [2] is chosen to be the data acquisition board for the prototype box. MDA320CA is a high-performance data acquisition board with up to 8 channels of 16-bit

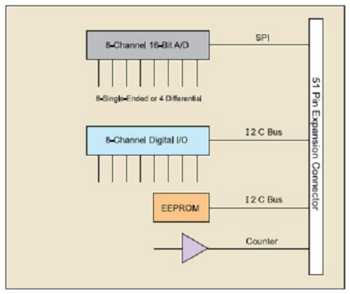


Figure 3.4: MDA320 Sensor Board

analog input. This board is designed for use in cost-sensitive applications requiring precision data collection and analysis. With improved micro-terminal connections, the MDA320CA offers users a rapid and convenient interface to a wide variety of discrete external sensing devices.

Analog sensors can be attached to different channels based on the expected precision and dynamic range. Signals with dynamic range of 0 to 2.5 V can be plugged to these channels. The analog to digital converter has 16-bit resolution. The least significant bit value is 0.6 mV. The result of ADC can be converted to voltage knowing that $Voltage = 2.5 * ADC_{READING}/65536$. Digital sensors can be attached to the provided digital or counter channels. Mote samples analog, digital or counter channels and can actuate via digital outputs.

MDA320CA sensor board is equipped with Texas Instrument ADS8344 analog-to-digital conversion chip. The ADS8344 is an 8-channel, 16-bit, sampling Analog-to-Digital (A/D) converter with a synchronous serial interface. Typical power dissipation is 10 mW at a 100 kHz throughput rate and a $+5$ V supply. The device includes a shutdown mode that reduces power dissipation to under 15μ W. Low power, high speed, and an on-board multiplexer make the ADS8344 ideal for battery-operated systems such as sensor measurement equipment for volcano monitoring. The serial interface also provides low-cost isolation for remote data acquisition. The ADS8344 is ensured over the -40C to +85C temperature range.

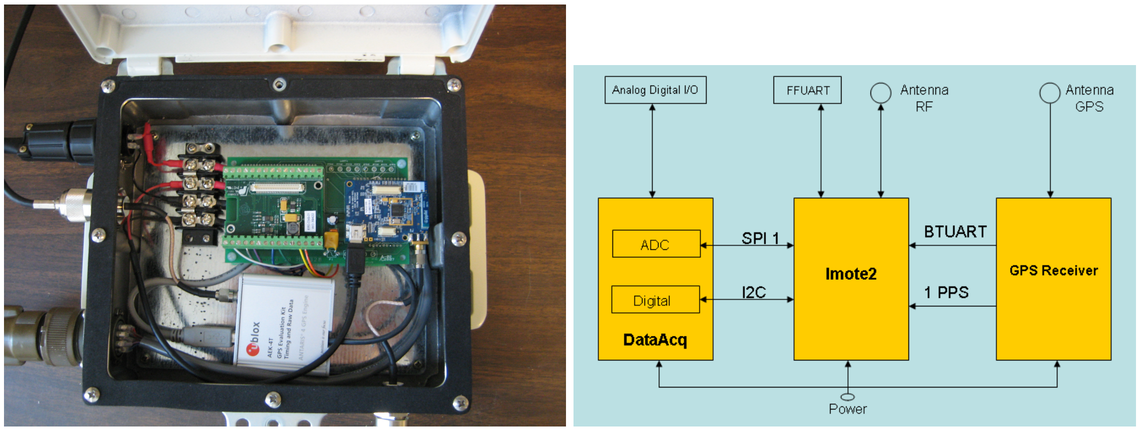


Figure 3.5: Prototype Sensor Box

3.7 Component Connection

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 3.5. iMote2 connects with MDA320CA sensor board through SPI interface 2 and the ADS8344 SPI clock signal is controlled by iMote2. The GPS receiver is connected to iMote2 through Blue Tooth UART interface, and GPIO 93 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 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 (Figure 3.5). Different components must be strictly ground power isolated and the electromagnetic interference problem must be avoided.

CHAPTER FOUR

CONFIGURABLE SENSING

In an environment monitoring application, different sensor nodes may be equipped with different type of sensors and need different data processing algorithms. Certainly, customized programs could be compiled and programmed for different stations. However, it is time-consuming and error-prone during deployment and is inconvenient for codes update over the air. The configurable sensor driver is highly desired in a system deployment: users can download the same program to different nodes and run online configurations after the deployment. This configurable feature includes both self-adaptive configurations and user-specified configurations.

4.1 Reliable Command Control

The online configuration is implemented through the RPC mechanism developed in [60] and the control command is carried out by the sink-initiated data dissemination. It is necessary to ensure 100% reliable delivery of such sink-initiated crucial control commands and the dissemination shall terminate in a short time period. Otherwise, it is a system failure from a user's viewpoint. In a feedback-driven monitoring sensor network [5], a connected science model may generate feedback commands to adjust network configurations in real-time. Those feedback commands have to be received by every sensor node in a right order with reasonable delay bounds. Motivated by those real system needs, the *Cascades* protocol is designed for reliable and fast propagations.

4.1.1 Reliable Broadcast Related Works

Much research has been done on broadcasting in wireless networks in the literature. Their major concerns include energy conservation and reliability. RBP [50] aims at providing adaptive reliability for broadcasts in varying network densities and improving the end-to-end reliability of flooding. It is distributed in the sense that every node makes its own decisions on retransmissions without any global or hard state. However, it does not offer 100% reliability guarantees. We borrow the idea of implicit ACK from it and use it to increase the Cascades protocol reliability.

Other broadcast protocols such as [8] focuses on minimizing the energy consumption of the nodes, through the minimization of the transmission cost of each broadcast in order to prolong the lifetime of the entire network. SmartGossip [26] is a probabilistic protocol that offers a broadcast service with low overheads. Smart gossip automatically and dynamically adapts transmission probabilities based on the underlying network topology. Its goal was to minimize energy consumption by reducing redundant transmissions. However, it cannot provide 100% reliability either.

Bayesian-Assisted Resource Discovery In Sensor Network, BARD, [49], uses stochastic processes to model both the searching and the routing components in the data dissemination process and it also uses prior history and the characteristic configuration of diffusion to avoid unnecessary flooding of related queries. When historic information is available, BARD is quickly and efficiently able to perform, however, BARD is limited by traditional flooding techniques when this information is not available. Spatiotemporal Multicasting or mobicast, [20], allows for the delivery of messages to a group of mobile destinations. In order to accomplish their goals of delivering the data to a mobile destination, it was necessary that the reliability constraints for mobicast be reduced.

[55] classifies existing broadcast schemes into categories and simulates a subset of each category, thus supplying a condensed but comprehensive side by side comparison. It determines the strength and weakness of each type as well as the situation(s) in which each is best suited. In [62], the author investigated and discovered tight bounds on both the broadcast capacity and the information diffusion rate in wireless networks for both dense and extended populations. $\Theta(log(n)^{-\frac{\alpha}{2}})$ was determined to be lower bounds of the broadcast rate for a continuous stream of data using a multihop implementation in extended networks. Furthermore, the authors also proved that constant diffusion, $\Theta(1)$, of data is possible in both dense and extended networks.

PSFQ (Pump Slowly, Fetch Quickly) [52] is a reliable transport protocol. It guarantees the reliability of downstream transmission by quick fetching from neighbor nodes after receiving packet with gap sequence from previous packets. When loss event is triggered, one NACK message will be broadcasted. However, PSFQ requires the sink node injected messages into network in a relative slow rate, pump slowly. The authors also indicate the importance of hop-by-hop error recovery where loss detection and recovery should be limited to a small number of hops (ideally one hop) due to the probability of losing packets accumulates exponentially with the number of hops. The NACK message is implemented as a request message in out broadcast mechanism.

GARUDA [41] uses Core-Recovery idea to implement reliable downstream data delivery. Some nodes in the network play the role as loss recovery servers, and other non-core nodes need to have one-hop connection with at least one core nodes. GARUDA works in two-stage recovery: in the first stage, downstream core nodes recover from upstream core nodes if detecting loss packet; in the second stage, non-core nodes recover from their neighbor core nodes.

Trickle, [28] uses a probabilistic or "gossiping" approach which utilizes meta-data in order to discover and repair missing data. Hence, theoretically, it eventually provides 100% reliability. But it might take a long time to converge. Deluge [21] builds off Trickle [28] and is the default code update protocol in TinyOS. In Trickle, nodes stay up-to-date by occasionally broadcasting a code summary to their neighbors. Trickle's key contribution is the use of suppression and dynamic adjustment of the broadcast rate to limit transmissions among neighboring nodes. While Trickle addresses single packet dissemination, Deluge extends it to support large data objects. However, according to [54], Deluge [21] is unable to provide reliable network programming in a field deployment.

4.1.2 Cascades Protocol

The *Cascades* protocol is designed for real environment monitoring sensor networks to achieve following goals:

1. To ensure that all commands are delivered to all the intended receivers with 100% reliability;

2. To ensure that a broadcasted session terminates in a reasonable short time period, instead of

gossiping forever [28];

3. To minimize the number of transmissions for lost detection and recovery operations with minimal signaling;

4. To operate correctly even in an environment where the radio link quality is poor;

In the protocol design, the network traffic and link quality is assumed to be dynamic: though nodes are not mobile, the communication links could be asymmetric and packet losses could happen due to poor quality links or congestions. The MAC protocol can be any existing MAC protocol as long as it supports broadcast. In the Cascades protocol, a data gathering routing protocol is assumed to be existed (such as MultihopLQI and Drain in TinyOS) and there are continuous data flow from individual sensor nodes to the sink node. Such a routing protocol only uses bidirectional links since the requirement of link-layer ARQ mechanism is not uncommon. Each node knows its parent for sure, and can learn its direct children to generate a children list by tracking recent forwarded packets. It is possible that the children list might be outdated if children switch parents without notifications.

Cascades Protocol Design

The Cascades protocol ensures 100% reliability by implementing parent-monitor-children analogy and reactive fetch mechanism [52] if there is a gap in sequences. The parent-children relationship is extracted from the data gathering tree on the fly. The essential algorithm works as follows:

1. Once a node u receives a new broadcasted message from another node v, it rebroadcasts if node u has children.

2. Node u retries periodically until it detects that all children have rebroadcasted the message or confirmed with an ACK.

3. If a node detects a gap in its received data sequences, it will broadcast a request (e.g., NACK) message to notify its parent.

Each node rebroadcasts periodically until successful receptions of all its children are confirmed either through rebroadcasts or explicit ACKs. If an asymmetric link occurs permanently (parent cannot reach children or children cannot reach parent), a correct data gathering routing protocol shall have updated the data gathering tree due to failed link layer acknowledgements. This is another reason why parent-children relationship is extracted from the data gathering tree, besides zero cost for dissemination. In the protocol implementation, a neighbor management module `NeighborMgmt`, which is connected to the routing module and learns parent-children relationships, is designed to perform such an extraction. In addition, the `NeighborMgmt` module snoops packets in neighborhood, discovers and maintains neighbors connectivity, hence it can detect node joining or leaving in a short time period and notify the `Cascades` module. With those supports, the `Cascades` module will be informed if a child joins/dies (e.g., update C_u) or the parent changes (e.g., processing parents switch event).

On the other hand, the Cascades protocol is a fast opportunistic data dissemination protocol. The broadcast flow does not depend on the data gathering tree structure and each node does not necessarily wait for disseminated messages from its own parent. A snooped new message from any other node will be accepted and rebroadcasted as well. The dissemination process is as fast as flooding. It is possible that a parent first hears new data from its child, before it hears it from the grandparent. For example, in Figure 4.1, node a is the parent of node b and f. However, node a's data reaches node b only, then b rebroadcasts the data to its neighbors (also as an implicit ACK to node a). At the same time, node h receives data from node l (though l is not node h's parent), and rebroadcasts the data to its neighbors which includes parent node f. The dissemination process therefore speeds up. Also, a non-leaf node may suppress its own re-broadcasting if all of its children have ACKed explicitly or implicitly, therefore saves communication costs. For example, in Figure 4.1, assume node a's DATA does not reach f, but node h's ACK and node g's DATA has reached f, then f only needs to send an ACK to a before a rebroadcasts. It is not difficult to imagine that the opportunistic design makes the protocol not just *faster*, but also more *robust*

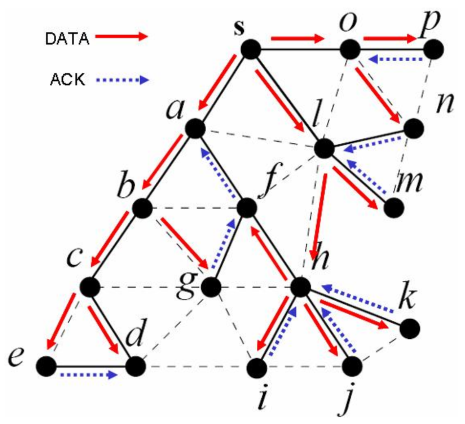


Figure 4.1: Opportunistic Broadcast Flow with Cascades Protocol.
The solid lines sketch the tree structure, and the dashed lines shows the other network links.

as a node does not solely rely on its parent node to get disseminated data. If the link reliability between a parent and child is temporarily bad, it still has a good opportunity to get the data from other neighboring nodes.

The protocol has been carefully designed to speed up disseminations, reduce communication costs, and increase the robustness. For instance, when node u receives a data message with sequence seq, it will add node x to its children list even if x is not node u's child. The reason is that: (1) in normal situation, it does not hurt to add a neighbor to the ACKed children list $C_u[seq]$, since u will determine whether it needs to rebroadcast or not based on the ACK status; (2) in abnormal situation, such that node x becomes a child of node u later than receiving the $DATA(seq,*)$ message, adding x to children list $C_u[seq]$ earlier will save some communication costs and increase the system robustness. Another example is that, no matter whether or not x is the parent of node u, as long as $u \in C_x^-[seq]$ (who has not received $DATA(seq)$ yet), node u will send $ACK(seq, parent)$ to node x. Notice that, this situation could happen if node u switches parent from x to its current $parent$. The message $ACK(seq, parent)$ will prevent node x from retrying forever (as it takes u as its child, while u does not takes x as parent). In addition, the message $ACK(seq, parent)$ also tells x that u's current parent has changed and x needs to perform corresponding actions.

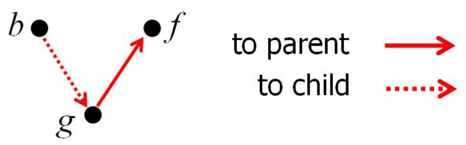


Figure 4.2: Possible Inconsistent Parent-Child Relationship during Route Updates

Theorem 1. *Cascades protocol provides 100% reliable data dissemination protocol from the sink to the entire network.*

Proof. The *Cascades* protocol achieves 100% reliability by implementing the parent-monitor-children analogy. In the data gathering tree, each node monitors whether or not its children receives the disseminated data. Each child node notifies its parent of successful reception through implicit or explicit ACKs:

1. Implicit ACK. Parent node snoops the packet from children. If a child rebroadcasts the same disseminated data, it implicitly means that the child has received the data already.

2. Explicit ACK. If the parent failed to snoop the implicit ACK, naturally, it will rebroadcast the data after the timeout period. Then, a child must reply with a short explicit ACK message to the parent node (if the child has previously received same data but is still in the parent's un-acked children list).

As long as any child fails to ACK implicitly or explicitly, the parent will rebroadcast periodically until every child has confirmed. Hence, reliability is guaranteed as long as the data gathering tree covers all nodes in the network and is consistent.

However, it is possible that, a node switches parents during a delivery session. Since every node maintains its own parent and children lists, a pair of nodes may have an inconsistent status. For instance, as shown in Figure 4.1.2, node b takes g as its child, but node g has already switched parents from node b to node f. The possible inconsistencies can be classified as one or both of the following:

1. Case 1: Node g takes node f as its parent, but node f does not take node g as its child.

2. Case 2: Node b takes node g as its child, but node g does not take node b as its parent.

Cascades overcomes these inconsistencies by having each node send a $REQUEST$ message to its new parent for missed data (if it exists) each time it switches parents. Thus, the inconsistencies of Case 1 are mitigated since each child explicitly asks for and is provided with missed data (if any exists) from its new parent. Also, node f adds node g to its list of children at this point.

For Case 2 inconsistencies, the only negative effect is that node b will retry even though node g is not its child.In Cascades algorithm, node b will terminate this retry after receiving $ACK(seq, f)$ from node g, as long as it happens. Node b then realizes that node g has switched parents to node f, and removes node g from its children list C_b. Notice that, as mentioned before, those temporary parent-children inconsistency will also be solved by the NeighborMgmt module. Hence, even if there are temporary asymmetric links (e.g., child cannot hear parent or parent cannot hear child), they will not retry forever.

In summary, *Cascades* protocol can handle those inconsistencies robustly to provide 100% reliability. $\square$

Cascades Implementation

In Section 4.1.2, the design principle of the Cascades protocol has been introduced. To implement it in a real system, some practical design issues shall be considered.

Nodes set data structure.

By sending ACK requested message, a node ensures that the child nodes in children list $C_x^-[seq]$ to implicitly or explicitly confirm the receipt. After been ACKed, the corresponding status of the child is set. The node id left $C_x^-[seq]$ will be added into the ACK requested message, and this message will be sent out after retry time our period. In fact, $C_x^-[seq]$ and $C_x[seq]$ are the same list in which ACK statuses are different for each node.

Timer implementation.

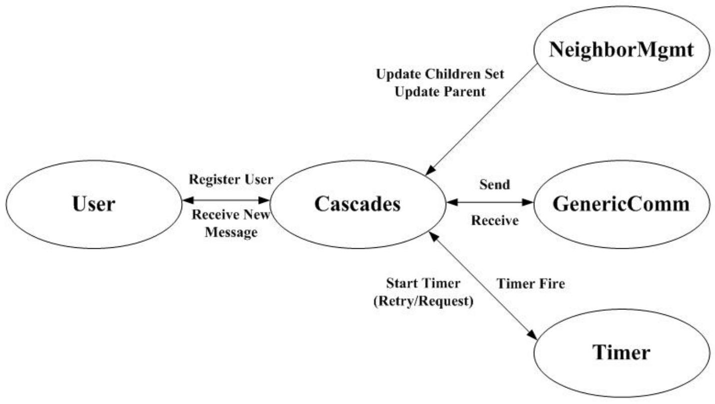


Figure 4.3: Cascades implementation in TinyOS

Non-leaf nodes must start a data resend timer $DT[seq]$ in order to wait for its children's acknowledgement of $DATA[seq]$. In the implementation under TinyOS, it is very resource expensive to start a timer for each seq. Instead, we define one repeat timer DT which fires every $t_0 = 50$ milliseconds, and define a timer counter array $DTC[\,]$ for all unacked packets. In other words, for packet $DATA[seq]$, $dt_{wait} = DTC[seq] \cdot t_0$. Once the timer DT fires, it scans the array $DTC[\,]$ and updates active $DTC[seq] = DTC[seq] - 1$: if $DTC[seq]$ reaches zero, it is equivalent to that the timer $DT[seq]$ fires. $DTC[seq]$ will be increased by one after receiving a duplicate packet from non-parent nodes and decreased by two after receiving a new packet to speed up disseminations.

Multiple clients and sinks support. It is possible that there are multiple users trying to send control commands to the network. In order to solve such a race condition and guarantee the increase of command sequence consistently, the command sequence is set just before commands are injected into the network. In addition, there might be multiple sink nodes in the network and the command distributing software needs to guarantee that all sinks confirm the injected messages. In current implementation, these two problems has been solved in the serial forwarder program which forwards network packets to the Ethernet and distributes commands for different clients to the sensor network.

4.1.3 Cascades Evaluation

The Cascades protocol performance is evaluated by running the OasisApp application in TinyOS. OasisApp is a multi-hop data collection application, based on a simple distance-vector routing protocol MultiHopLQI. Each node periodically samples data and forwards it to a gateway; hence the data flow forms a sink tree rooted at the gateway. A NeighborMgmt module which maintains the neighborhood connectivity and parent-children information is connected to it to leverage the existence of MultiHopLQI (See Figure 4.3 for illustration). In Figure 4.3, CascadesM is the main module for Cascades: all broadcasted data packets and control packets are processed in this module. GeneriComm is the connection between CascadesM and the default TinyOS MAC module which is a CSMA based MAC protocol.

The testbed experiments were performed in an indoor sensor network of 15 iMote2 motes. The CC2420 radio power of iMote2 was set to power level two which is still capable of forming a multi-hop network. In the experiments, PC injected one packet into the network every τ time interval and the experiment results showed that Cascades performs better in all three of the evaluation considerations than Drip.

Drip is a transport-layer component in TinyOS based on Trickle [28] for epidemically disseminating messages throughout the network. After sending a message, Drip will continuously retransmit to ensure that it eventually reaches every node in the network. Drip will delay its own transmission after receiving duplicate packets from neighboring nodes.

The following metrics are used to evaluate the performance of Cascades Protocol:

1. Reliability, which measures the number of received packets to the number of sink-injected packets in different time checking points. $P_{recv}[i]$ is the number of received packets in node i. In each checking point, the reliability is defined as $\frac{\sum_{i=1}^{N} P_{recv}[i]}{N \cdot Pkt}$. N is the number of nodes in the network and Pkt is the number of injected packets, so the desired total received packets number is $N \cdot Pkt$.

2. Communication Cost, the average number of broadcasted data packets in the network, which is $\frac{\sum_{i=1}^{N} P_{sent}[i]}{N}$. N is the number of nodes in the network and $P_{sent}[i]$ is the number of broadcasted data packets in node i. Communication cost can also represent the energy consuming in the network.

Reliability Evaluation

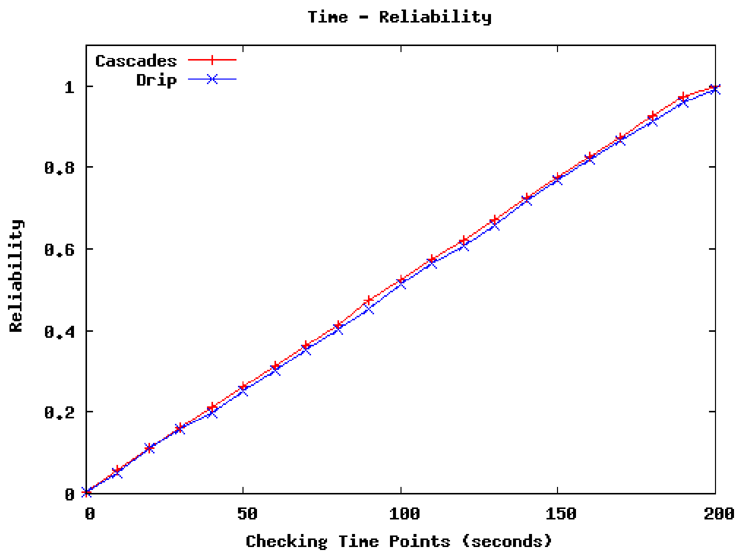


Figure 4.4: Reliability in Different Checking Time
X axis is checking time points in second unit and Y axis is reliability value. Number of nodes is 15, Pkt=1000, $\tau = 2000\ ms$.

From Figure 4.4 we can see that the average reliability of Drip is 99.8% while it is 100% for Cascades. This means that the whole network lost 30 packets with 1000 sink injected packets running in Drip. The performances of Drip and Cascades are similar in most of time, but after losing several packets, the reliability of Drip fell a little bit. Such packet lost events could be explained as: 1) in some cases, Drip cannot guarantee all nodes in the network to receive sink-injected packet within τ period, which is $2000ms$ in our experiment; and 2) during experiments, it is also noticed one or two packets may be intercepted in the sink node. Though such event does not always happen, it is a system problem from a user's viewpoint.

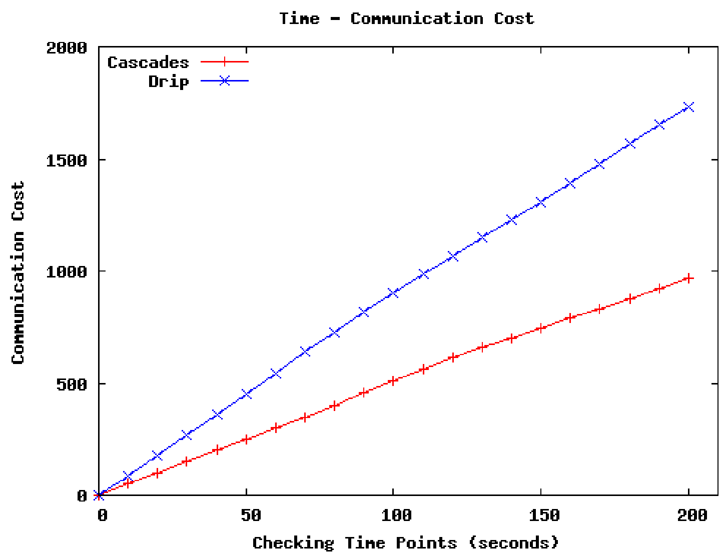


Figure 4.5: Communication Cost in Different Checking Time
X axis is checking time points in second unit and Y axis is communication cost. Number of nodes is 15, Pkt=1000, $\tau = 2000\ ms$.

Communication Cost Evaluation

In the testbed experiment, it only cost Cascades 14505 packets to finish converging on average, yet Drip spent 26025 packets. For each node, the average broadcasted packet is 967 for Cascades and 1735 for Drip (see Figure 4.5). There are three reasons lead to the drop of communication cost in Cascades: 1) the leaf node will not broadcast after receiving packets from the parent node; 2) the implicit/explicit ACK mechanism reduces the transmission in parent nodes; and 3) there is no meta-data message exchange between PC and the network to query the current command message sequence.

Time Delay Evaluation

The message delivery delay is also evaluated. For each packet, the interval between the last receiving time across the network and the sink injected time is recorded. The average delay time for all packets in the network is 303.4 millisecond for Cascades and 344.1 millisecond for Drip. From Figure 4.6 we can see that the delay value between Cascades and Drip is close to each other for

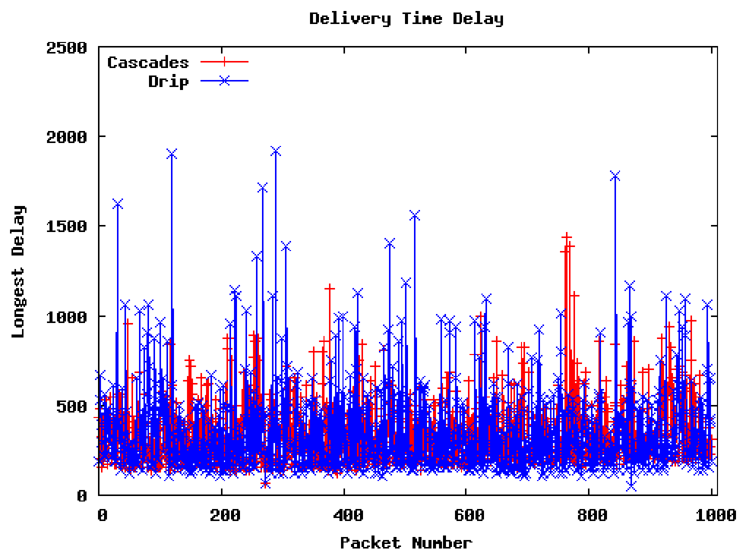


Figure 4.6: Longest Time Delay for Each Packet
X axis is packets sequence injected by sink node and Y axis is the longest delivery time delay in the network. Number of nodes is 15, Pkt=1000, $\tau = 2000\ ms$.

most packets. However, there are more packets have longer delay (over one second) in Drip due to the different retry mechanism.

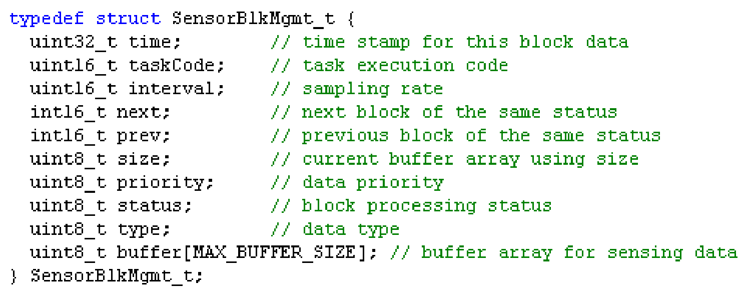
4.2 Configurable Sensing Services

4.2.1 Parameter Tuning

The sensing module in Figure 2.2 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 and reference voltage. All of these parameters could be adjusted based on the system status to improve system performances and data fidelities. For example, the seismic sensor priority could be increased to allocate more bandwidths for seismic data than other type of sensors when an earthquake event is detected.

Parameter tuning provides a service of updating sensing related parameters under a user's requirements.

- Sampling Rate: The sampling rate of each sensor channel is related to the whole system: a



```
typedef struct SensorBlkMgmt_t {
  uint32_t time;         // time stamp for this block data
  uint16_t taskCode;     // task execution code
  uint16_t interval;     // sampling rate
  int16_t next;          // next block of the same status
  int16_t prev;          // previous block of the same status
  uint8_t size;          // current buffer array using size
  uint8_t priority;      // data priority
  uint8_t status;        // block processing status
  uint8_t type;          // data type
  uint8_t buffer[MAX_BUFFER_SIZE]; // buffer array for sensing data
} SensorBlkMgmt_t;
```

Figure 4.7: Memory Block Data Structure

higher frequency sampling rate generates more data, uses larger memory space for storage and possibly occupies the network. By setting different sampling rates at different time, a system is able to be adaptive to network situation variants. The sampling rate is one of the most frequently updated parameters.

- Data Priority: In different time of periods, different sensor data may contain variable levels of information. Scientists may raise or lower the priority of different data to help collecting more user-concerned data under non-reliable network status. Higher priority data are allocated with more transmission opportunities, which would help to increase the data fidelity and yield from a user's perspective.

- Node Priority: A sensor node in different area of the network may have different weights according to temporal and spario situations. E.g., a node more close to eruption spot is more important than other nodes. By assigning high node priorities to those nodes and combining the node priority and data priority would increase the fairness of data priority.

It is also possible to add other types of parameters with the help of RPC mechanism [60].

4.2.2 Memory Management

The sensor memory management module is developed to save sensed data into different storage mediums based on platform attributes and network situations. If there is enough free RAM space to use, the sampling data will be saved into RAM space instead of Flash chip. However, network congestions or other problems may cause a tight memory resource condition. After detecting such a situation, the memory 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 the iMote2 node, it is efficient to allocate a big buffer space in RAM for sensing data management and move old data into Flash chip only if the memory usage level is over a warning threshold. In general, 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 a TinyOS system, memory shall be statically allocated during compilation time, as it does not provide safe dynamic memory management mechanism. Hence, it causes a lot of inconvenience especially when we need to design configurable sensor drivers. We cannot predefine the memory block size for each sensor, since we even do not know how many sensors would be connected and what their sampling rates during compilation time. To solve this problem, the sampling data is segmented into fixed-size data blocks which are then managed through a queue management module. Each of the data block (Figure 4.7) has the information of the sampling start time, the sampling rate, the data processing task code, the block 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 enough sample data, the status will change to a pending status which is recognized by different processing tasks (will be discussed in Section 4.2.6). A sensor block will be freed after it has been processed successfully.

Besides saving data into different storage mediums, the memory management module is also in charge of managing memory space based on sampling rates and network situations. Different sensors may run with different sampling rates in a system, e.g., seismic sensor sample at 100 Hz while lightning sensor samples at 1 Hz. It is not fair to let seismic sensor seize all memory space because of more opportunities to get memory resources. For each type of sensor, the maximum block number shall be decided by the sampling rate, in which case, a lower sampling rate sensor needs to be guaranteed to have blocks to save data. If the allocated memory block number is over the allowable maximum value, a sensor can only get new blocks by replacing older blocks of it. The maximum block number of different sensors also needs to be adjusted according to sampling rate changes.

4.2.3 Read/Write Parameters into Flash

In case of system error occurrences, the enabled watchdog will reboot the node in order to drive it back to the normal running mode. However, the previous settings will be lost if all configuration parameters are saved in RAM space and the whole network will work in an inconsistent status. Saving parameters into flash storage component is necessary to guarantee system consistency.

When operating the flash write/read command, we need to format the flash at first. The reason is because that the flash space is not only used for parameters storage, but also used for wireless programming that will save updated code segments (golden image in flash) before rebooting to a new program. The flash component is also used to back up sensing data in case of long period network congestion, a bad radio condition, etc.

In the current implementation, all sensing parameters are saved into the first block in the flash chip and the first byte in the starting address is a flag. If the flag is set, the sensing component will be initialized by the value in the flash, otherwise, it is initialized by default parameters. Every time the sensing parameters are changed, a flash writing task is posted.

4.2.4 Generic Support to Different Platforms

Sensing component is not only designed for the OASIS system. In order to make it reusable and generic, we need to design generic APIs and hardware abstraction modules to support other applications. For instance, the Real Time Clock module will only provide one universal interface to up layer as Interface RealTime, and it hides the low level information like the clock fire mode, interval. The sensing component will only call the universal interface $ADC.getData()$ to sample sensor data. What's more, for a GPS receiver, it generally have two interfaced been used, the GPIO interrupt interface and the UART communication interface. We just provide general interfaces like `getGPSData`, and `signalGPSData` to the up layer.

For different platforms, the MICAz mote has an 8-bit CPU, the Tmote Sky sensor node has a 16-bit CPU and the iMote2 node is driven by a 32-bit CPU. Tmote2 Sky has an internal ADC component, while MICAz and iMote2 has to be connected with an external sensor board to do sensing. In addition, the sensor board connected with MICAz is through I2C interface while the iMote2 node needs to communicate with the sensor board followed the SPI protocol.

4.2.5 Configurable Sensor Selection

Even though all the nodes in the network are running the same application software, they could dynamically add or delete some new sensor type 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 to the node later and it is not necessary to reprogram the software to accommodate to such a change. For example, if users are willing to add a new sensor like an SO_2 sensor to detect the air condition when eruption happens, and we just need to plug the new sensor in, and send one command to select the corresponding analog channel. If some type of sensor in the node is declared to be broken, we just need to delete it and free the allocated buffer. As discussed in Section 4.2.2, our design enables scalable memory allocation for each sensor to maximize the memory usage.

4.2.6 Configurable Processing Task Schedule

In order to apply different data analyzing algorithm to sensed data, we designed each processing algorithm as a function and put the according pointer into a processing table, and the task code of each function is the index in that table. The function prototype is defined like:

$result\_t(*ProcessFunc)(SenBlkPtr\ inPtr,\ SenBlkPtr\ outPtr)$

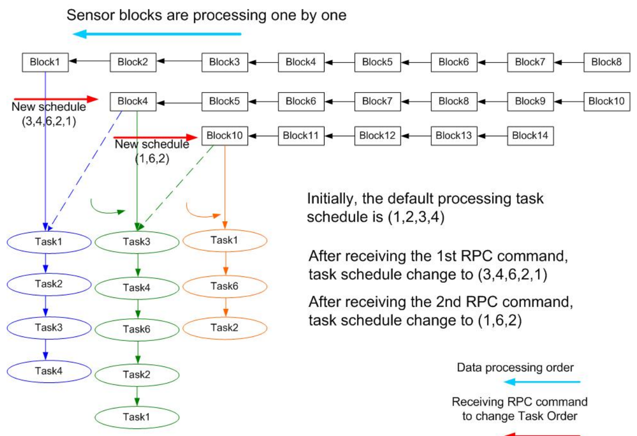

Sensor blocks are processing one by one
Block1
Block2
Block3
Block4
Block5
Block6
Block7
Block8
Block9
Block10
Block11
Block12
Block13
Block14
New schedule (3,4,6,2,1)
New schedule (1,6,2)
Task1
Task2
Task3
Task4
Task6
Initially, the default processing task schedule is (1,2,3,4)
After receiving the 1st RPC command, task schedule change to (3,4,6,2,1)
After receiving the 2nd RPC command, task schedule change to (1,6,2)
Data processing order
Receiving RPC command to change Task Order

Figure 4.8: Processing Task Scheduling

The two parameters, $inPtr$ denotes the sensor data block to be processed, and the $outPtr$ denotes the output block.

Similar as a configurable sensor driver, users may need some flexibility to specify the corresponding processing tasks for different type of sensors. Again, it is preferred to be online configurable, so that users can change the execution sequence of different 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 [40], which is the well-known seismic signal processing algorithm, and the other is a simple threshold 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 have been programmed into the node and the default data processing task list is $(1, 2, 3, 4)$. This list could be configured to $(3, 4, 6, 2, 1)$, $(1, 6, 2)$ or any other permutation, see Figure 4.8. In Section 4.2.2, 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 4.7), 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. With the support of the incremental reprogramming over the air tools, a new task function's binary code could be sent to the node to extend the existing processing tasks. It would be even more scalable and configurable.

Currently, several processing tasks have been developed.

- RSAM calculation task: this task outputs the average value of sensing data in one second period. It needs to set the output pointer to a new sensor block.
- StaLta task: this task compares the short term average value to the long term average value to detect the seismic event (see Section 5.2.2).
- Compression task: this task compresses the input data. The output of this task is set to a new sensor block.
- Prioritize task: this task changes the priority information of each data block depends on the event detection result.
- Threshold task: this task is used to detect the lightning event and it increases the priority of the lightning data if a strike is detected.

CHAPTER FIVE

SITUATION AWARENESS

The processing component in Figure 2.2 implements several situation awareness based analyzing algorithms to optimize network resource usages under dynamic environment situations. In current smart sensing design, it includes error detection and recovery, event detection and data prioritization. This module could also configure other modules adaptively based on analyzing the result. In order to implement correlated event detection, fault tolerance and data aggregation, the processing module needs to overhear neighbor data messages.

5.1 Event Detection Related Works

Event detection based on threshold comparison is the most widely methods, and there are a number of sensor database systems implemented recently based on event detection oriented query processing. The Cougar [58] system encapsulates the logic of threshold-based event detection into asynchronous functions in the system. TinyDB [31, 32] processes events generated from a threshold-based query or a module hand-coded in the operating system. However, both of these two systems miss the join operation. REED [7], on the other hand, extends TinyDB to support efficient in-network join operations, which are useful for event detection in industrial process control applications. In the smart sensing, a simple threshold event detection function is implemented serving as a basic detection method.

Other event detection methods like [17] is the mechanism based on learning of time series data. This approach allows users to detect a change-point by detecting the change of model that describes the underlying data. The author combined the change point detection and model selection techniques as a hybrid mode of detection approach, and this approach does not assume the availability of points in the time series and it is independent of regression and model selection methods. Such time stream data mining scheme could also be applied to sensor network systems.

In [63], the SAX [29] 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; and 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 big storage space in their approach is not feasible for high data rate sensor network systems.

Though the time-streaming mining method is a powerful event detection method, it required high computation capabilities and memory usages. The STA/LTA event detection implemented in smart sensing component is lightweight compared with those two methods and the accuracy is desirable in environment monitoring data analyzing.

The system in [35] 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 design also uses similar idea of alarm report, but we provide the configurable alarm report mechanism instead of using the correlation information from neighboring nodes.

Contour map event detection proposed in [57] is an event detection mechanism based on matching the contour maps of in-network sensory data distribution. Because the changes in the sensor readings of networked nodes that are caused by an event usually exhibit some spatio-temporal pattern in some application like a coal mine emergency guiding system, the spatio-temporal pattern recognition event detection is better than simple thresholds detection in such a system. [57] 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 an event detection approach is useful in querying the network, however, it is not feasible for real-time sensor network due to high computation and communication costs.

[61] talks about an automatic P-wave arrival detection and picking algorithm based on Akaike information criteria (AIC) calculation on the wavelet transform of seismogram. Traditionally, when AIC picker is applied to the seismogram, the P-Wave arrival time is the point where the AIC has a minimum value. However, the signal to noise ratio in the seismogram affects the accuracy of the AIC picker to some extent. By applying AIC picker directly to the absolute wavelet coefficients in different scales, it could greatly reduce the affect of random noise or other incoherent features. If the picks at three scales are consistent, and then a P-wave arrival is declared. It requires an integer based wavelet transformation in order to implement such a method inside a sensor node, and some denoising techniques also need to be integrated.

5.2 Environment Event Detection

Distributed event detection is an important feature in smart sensing design: it could enable efficient node resource utilization and high fidelity network reconstruction in the base station. The simple detection 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,the 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 [40] to locate the earthquake event period in each sensor nodes. STA/LTA is a general event detection algorithm in seismic data analyzing.

5.2.1 Threshold Event Detection

Threshold based event detection is an easy implemented detection scheme. It is useful in several scenarios, such as temperature overflow, lightning strike, etc.

The amplitude threshold trigger or the level trigger simply searches for any amplitude exceeding a preset threshold. Recording starts whenever this threshold is searches and stops when the level is below the threshold or after a given time. This algorithm is often used in strong motions seismic instruments, where high sensitivity is not an issue. The level trigger is now largely replaced by the STA/LTA trigger; however, many instruments still keep the level option for specific applications.

In the current smart sensing design, the threshold trigger event detection is implemented as one signal processing task among other processing tasks. It is applied to the lightning sensor data which always has "strike value" when lightning happens, and that strike value would be set as a threshold which is much higher than general data.

5.2.2 STA/LTA Event Detection

The STA/LTA event detection mechanism is a mature volcano signal processing algorithms [40], and the RSAM - Real-Time Seismic-Amplitude Measurement is calculated on one second continuous sensing data and saved into STA/LTA buffer for event detection usages.

In the smart sensing implementation, continuous sensor blocks with seismic RSAM data are processed by following equations:

$$X_i = \frac{\sum_{j=1}^{n} x_{i-j}}{n} \tag{5.1}$$

$$X_i = X_{i-1} + \frac{x_i - x_{i-n}}{n} \tag{5.2}$$

where x_i is the RSAM value of sensing data in one second period; n is the STA or LTA windows in one second unit; 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. During the implementation, we experiments several trigger parameters and their settings.

STA time is the window length over which STA is calculated. Values can typically be from 0.3 second to 0.5 second for local earthquakes. A short STA time makes the trigger more sensitive to short term variations in the signal, while a longer STA is better at averaging out short term fluctuations and it can be compared to a low pass filter. In the OASIS system, STA is selected to be 8 seconds: if there are spikes in the signal, a longer STA must be used in order to average out the interference. This implementation will reduce the sensitivity to short lasting signals, but if a longer lasting signal is the objective, STA can be as long as the duration of the main P-wave trains.

LTA time is the window length over which LTA is calculated. The main objective is to have the LTA short enough to adapt to slow changes of the background noise and long enough to avoid reducing the sensitivity to triggering on low amplitude emergent signals. Since the natural background noise usually changes every slowly, it is generally better to use a too long than a too short LTA-time. In the OASIS system, the LTA value is set to 30 seconds for a higher sensitivity: such a carefully selected LTA-time might prevent triggering on slowly emergent man made signals like a truck passing, which typically can last for 2 minutes, while still retain sensitivity to more impulsive signals. However, the tradeoff is that it might also prevent triggering on other real signals.

Freezing LTA is the option of keeping the LTA at its pre-trigger level during detection. The advantage is that STA/LTA will reflect the true STA/LTA of the earthquake signal during the vent detection and the trigger will terminate at the true de-trigger ratio. If the LTA is not frozen, it is continuously increased and the de-triggering will occur too early, but a long LTA-time can partly prevent this. In our experiment, a sudden increase in background noise that triggers the system will keep it in trigger state forever and we chose not to freeze the LTA window to avoid such a disadvantage.

Event triggering starts when STA/LTA exceeds the minimum trigger ratio or trigger threshold value. The higher the ratio is set, the fewer events are recorded. A too low value results in many false triggers. So the minimum trigger ratio is one the most important parameters to set for optimum performances. The ratio can be lower at low noise sites (meaning few impulsive disturbances). During the development, we tried ratio 3 and 4 which resulted in insensitive event triggering. A high ratio is always excepted used for strong motion recorders, which often are in noisy places and only need to trigger on strong signals. Current smart sensing design uses a short STA window with a low ratio 2 to be sensitive to event changes.

Pre-event memory time is the number of seconds the recording starts before the trigger time. We configured the pre-event memory time to be 60 s for regional earthquakes in which S-P time is assumed to be 30 s. Since data is digitally recorded, there is always a buffer in memory of past data, so, when the event is identified, the recording starts earlier that the trigger time. Even for the most impulsive signals, the trigger time will be delayed relative to the very first onset, so if recording started at the trigger time, the initial onset would be lost. For emergent signals, the initial trigger might be several seconds into the record or as late as the S-phase. There is furthermore a need to have a reasonable noise record in front of the first onset in order to evaluate signal to noise ratio. Thus the value should be large enough to get the P-phase and a noise sample in front if the triggering is one the S-phase.

5.2.3 Data Prioritization

In smart sensing design, different priorities are assigned to different types of data based on the event detection result, and the communication stack will give more transmission opportunities to the higher priority data. 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

Sensor Name	Data Name	Data Type Value	Data Priority Vale
Seismic	RSAM1: Seismic RSAM Value	1	7
Seismic	Seismic Triggered Events	2	5
Seismic	Seismic Continuous Data	2	2
Infrasonic	RSAM2: Infrasonic RSAM Value	3	6
Infrasonic	Infrasonic Triggered Events	4	4
Infrasonic	Infrasonic Continuous Data	4	1
Lightning	Lightning Strike Data	5	6
GPS	GPS UBX-RXM-RAW Data	6	3

Table 5.1: Data Type and Priority

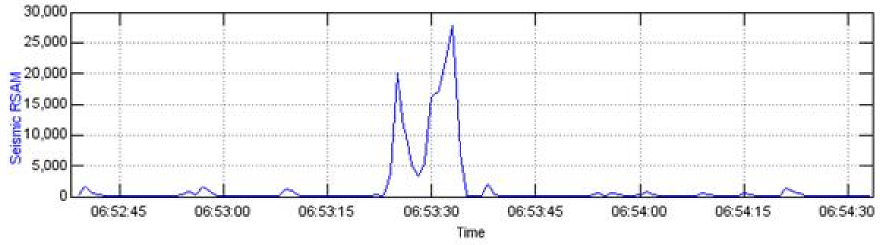
priority messages.

Though only seismic data is applied to the STA/LTA event detection algorithm, both seismic and infrasonic data are processed by the prioritizing function. Data sets two seconds ahead of event start time and one second later than the event end time are viewed as high priority data.

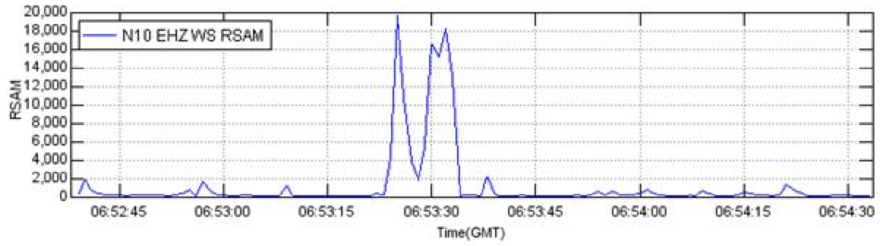
Table 5.1 shows different sensor types and their corresponding priority values. Both seismic and infrasonic data have three types of data: RSAM data which is the average value per second, continuous data which is non-event data and triggered event data which fall into the triggering window mentioned in section 5.2. The triggered event data and continuous data share the same sensor type value with different priority value. Table 5.2 lists the priority value ranges from 7 (highest priority) to 1 (lowest priority). The default priority of seismic is 2 and 1 for infrasonic data, and the priority of event periods data is increased by 3. Different from raw data, both RSAM values are viewed as high priority data. The priority of GPS data is only assigned for 3 due to its characteristic of fewer changes.

5.3 Event Detection Evaluation

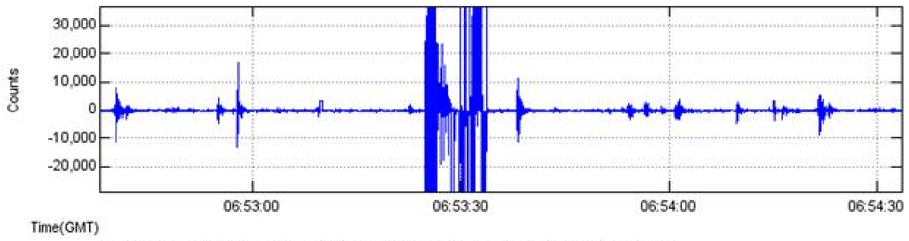
In Figure 5.1 (a) and (b), the node software calculated RSAM value (send back to control center through radio) and VALVE system calculated RSAM value (PC calculated value based on raw monitoring data) are shown to match closely, thus validity of the event detection algorithm running



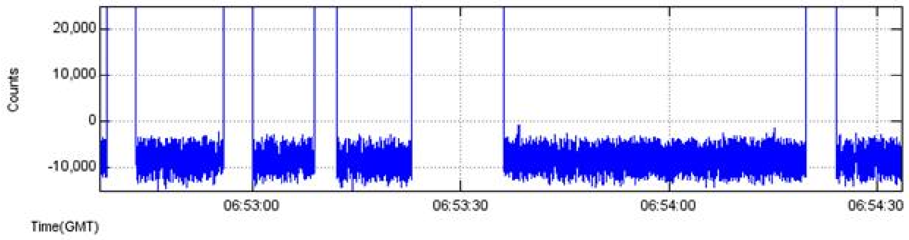
(a) Node Software Calculated RSAM Value



(b) VALVE System Calculated RSAM Value



(c) Seismic Data Channel with Real Volcano Monitoring Data



(d) Modified Infrasonic Data Channel with Event Period Marked

Figure 5.1: Event Detection Result

Data Priority	Data Type
7	RSAM1 seismic
6	RSAM2 infrasonic + Lightning
5	Triggered Seismic Events
4	Triggered Infrasonic Events
3	GPS UBX-RXM-RAW Data
2	Seismic Continuous Data
1	Infrasonic Continuous Data

Table 5.2: Priority

inside a sensor node is guaranteed. In order to show visualized event detection result, the infrasonic channel data was changed: If an event is detected by the STA/LTA algorithm on the seismic data channel (the input data is real volcano monitoring seismic data), the corresponding infrasonic data in the same event time window is set to the highest value as shown in Figure 5.1 (c) and (d), otherwise, the infrasonic data is random value. From Figure 5.1 we can see that all those four waveforms matched with each other. The event start time is set to two seconds before the trigger time and the ending time is set to one second later than the de-trigger time.

5.4 Fault Detection and Recovery

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

- Sensor board disconnection. Due to the unstable sensor board connection, the sensing value would always be $0xFFFF$. By analyzing the maximum, minimum and average value of sampling data, a sensor board disconnection problem would be detected when all those three values are equal to $0xFFFF$.
- Broken sensor. A broken sensor error could be caused by sensor disconnections in which case the ADC channel might appear as if it was sampling random data; sensor damage that

might present a symptom of the environment changes on the data is also one type of broken sensor error. The predefined error pattern to detect such errors could be mixed up with the background noise and lead to a false detection. A complex pattern recognition algorithm or collaborative error detection mechanism could be applied to identify the broken sensor.

- System alarm detection. System alarm includes low battery voltage, low free memory space and other serious situations for the sensing component. In the OASIS system, a batch timer is started to check battery, memory, radio or other components to update the alarm signal.

CHAPTER SIX

SYNCHRONIZED SAMPLING

Accurate timing of recorded signals is paramount when analyzing volcano monitoring data. Also, correlating signals across the sensor array requires accurately time-stamping each sample. Besides the requirement of accurate time-stamp, volcanologists also require that different nodes sample the same sensor channel in synchronized fashion, with time differences less than one millisecond. That is to say, when node u and v are sampling the seismic data in 100 Hz, they shall sample at the same millisecond in every 10 milliseconds.

6.1 Time Synchronization Related Works

Most data loggers provide the ability to time-stamp samples collected by different sensors. Samples need to be accurately time-stamped in order to correlate readings from different sensors. Even if the data sets are used only for frequency analysis, it is necessary to times-tamp samples in order to distinguish responses due to different events. Most of the proposed times stamping schemes address pair-wise clock synchronization between nodes, and network-wide synchronization to a reference clock.

The Network Time Protocol (NTP) [38] is a protocol for synchronizing the clocks of computer systems over packet-switched, variable-latency data networks. NTP is one of the oldest Internet protocols still in use, and it stands out by virtue of its scalability, self-configuration in large multi-hop networks, and robustness to failures. However, NTP is not feasible for large-scale networks of small, wireless, low-power sensors nodes. The WSNs require that time be synchronized more precisely than in traditional Internet applications due to their close coupling with the physical world and their energy constraints.

In Reference-Broadcast Synchronization (RBS) [14], nodes send reference beacons to their neighbors using physical-layer broadcasts. Such a reference broadcast does not contain an explicit timestamp; instead, receivers use its arrival time as a point of reference for comparing their clocks. 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 [16] algorithm performs pair-wise 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 [34] has been successfully integrated into some existing WSN systems [54]. 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 time-stamping and comprehensive error compensation including clock skew estimation, FTSP reached high precision performance. 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.

In Wisden system [56], 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. 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 to each packet) 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,

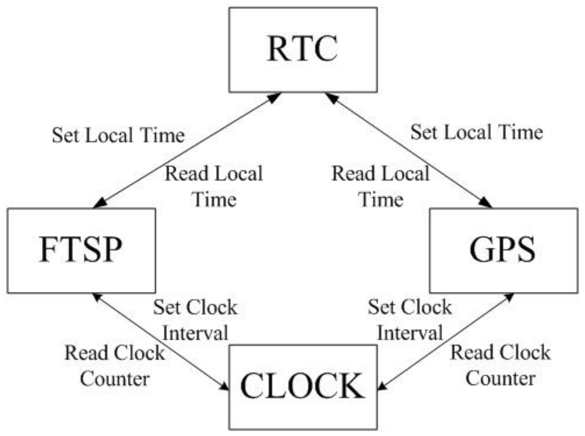


Figure 6.1: Synchronization Process

the more the drift affection is. In both FTSP and Wisden, the propagation time is neglected.

Even though that RBS, TPSN, FTSP, and Wisden have been implemented to support accurately time-stamping data, none of them proposed the solution to the network-wide synchronized sampling problem which is one of the contributions of the smart sensing design.

6.2 Synchronized Sampling Implementation

The synchronization mechanism in smart sensing design provides both services of accurately time-stamping and synchronically sampling among the network.

Figure 6.1 depicts the synchronized sampling framework. The timer event from RTC module drives all nodes in the network to conduct sampling synchronically, and the time-stamp is acquired from the local time value maintained in RTC module. The essence of the synchronous timer and accurate time-stamp is guaranteed by the time synchronization module, which could either be FTSP or GPS.

6.2.1 Real Time Clock Module

The RTC module (Figure 2.2, 6.1) maintains a millisecond resolution 32-bit local time value, which is updated by 1 in each clock fire event signaled from a low level clock. The default fire interval of this low level clock is configured to be 1000 microseconds to guarantee the updates of RTC time in every millisecond.

Though the pulse-per-second signal generated by the GPS receiver provides a one second resolution accurate synchronized event, the high frequency sampling needs a ten millisecond fire interval at least. Therefore, it is a must that the RTC module needs to provide one generic `Timer` interface to satisfy the requirement of synchronized sampling.

The sensing module uses the RTC module's timer service to generate short interval synchronized events, and the fire interval of this timer is the greatest common devisor of all sensors' sampling rates. Assuming all 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 calls timer interface provided by RTC module to start sensing with a fire interval δ; and 2) assuming current time is $T_{current}$, the fire point T_{fire} is set to $\lceil \frac{T_{current}}{\delta} \rceil * \delta$ instead of $T_{current} + \delta$, and an event is generated when $T_{current}$ matches T_{fire}. For example, $\delta = 10ms$, $T_{current}$ is `20:00:00:422`, then the next fire interval is `20:00:00:430` but not `20:00:00:432`. After setting T_{fire} for the first time, the successive T_{fire} is updated by adding δ to the last T_{fire} to enable continuous event signals. The time-stamp for each sensor block (Figure 4.7) is the first data sampling time of that block.

Such a synchronized timer service also provides an opportunity for other components requiring time synchronized behavior such as a TDMA MAC layer implementation.

The RTC module also provides a simple `RealTime` interface to up layer components, which has following groups of functions:

- Time information query functions, `getMilliSecond()`, `getSecond()`, `getMinute()` and `getHour()`, each of which returns the corresponding time information as required; `getTime(RTClock_t RTtime)` sets the time structure with corresponding time information.

- Time adjustment functions, `getTimeCount()` which returns the local time value, `setTimeCount(uint32_t count, uint8_t mode)` which is called to set the local time after getting the correct time from the synchronization module; and `setOffset(int32_t offset)` is used to adjust the low level clock fire interval.

- Synchronization mode functions, `isSync()` is used to check whether the local time has been synchronized with the global time, and `changeMode(uint8_t mode)` is called either by FTSP or GPS module to set the time synchronization mode.

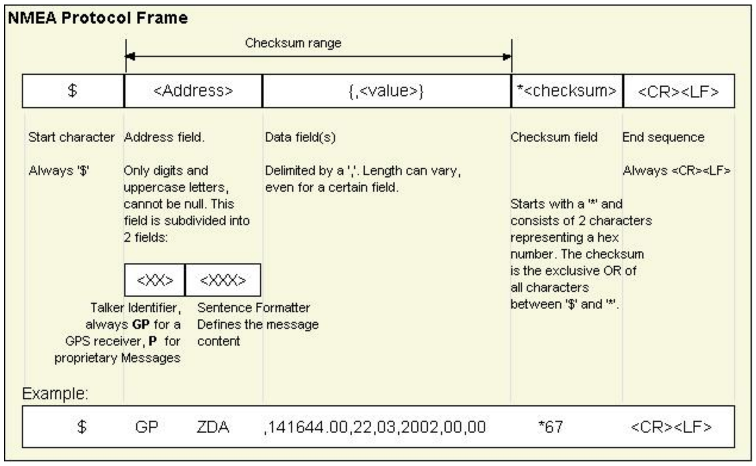
6.2.2 Microsecond Resolution Clock

The low level clock that is used to update RTC local time every one millisecond is the Operating Clock of Channel 10 on the iMote2 platform. The clock counter register will be updated by hardware per clock tick and a fire event is generated after matching between the clock counter register and match register. The clock frequency of channel 10 could be configured to 3 Mhz, 1 Mhz, and 1 Khz. In the current implementation, it is set to be 1 Mhz that means 1 microsecond per clock tick. The reason of such a configuration is to provide an easy way to adjust the clock fire interval with a finer resolution based on the time synchronization result, and it will be discussed in Section 6.3.

Ideally, each clock tick is assumed to be 1 microsecond; however, the real value is not that perfect: a clock running in 1 Mhz on iMote2 platform elapses about 27 ticks per second in experiments, and the value varies on different nodes due to crystal ageing differences. If the clock fire interval is set to be 1000 ticks (assumed to be 1000 microsecond), $27 * 60 = 1620$ ticks will be lost in every minute, and it will make the local time count value to be 1 millisecond less than the real time. The time synchronization mechanism in Smart Sensing design successfully fixed this problem.

6.3 Time Synchronization

Time synchronization undergoes several periods during system running. The first period is the system boot-up time during which there is no GPS signal received and there is no synchronization



NMEA Protocol Frame
Checksum range
$
<Address>
{,<value>}
*<checksum>
<CR><LF>
Start character
Always '$'
Address field.
Only digits and uppercase letters, cannot be null. This field is subdivided into 2 fields:
<XX>
<XXX>
Talker Identifier, always **GP** for a GPS receiver, **P** for proprietary Messages
Sentence Formatter Defines the message content
Data field(s)
Delimited by a ','. Length can vary, even for a certain field.
Checksum field
Starts with a '*' and consists of 2 characters representing a hex number. The checksum is the exclusive OR of all characters between '$' and '*'.
End sequence
Always <CR><LF>
Example:
$ GP ZDA ,141644.00,22,03,2002,00,00 *67 <CR><LF>

Figure 6.2: NMEA Message Format

actions performed. The second period is to select between GPS or FTSP synchronization mechanism, and the FTSP mode will be selected if a node cannot detect valid GPS signal for twenty seconds after rebooting. The third period is to keep updating RTC time according to the synchronization result. If the GPS signal is lost or recovered, the fourth period would be switching between the two time synchronization mechanisms.

6.3.1 GPS Data Format

Message	**RXM-RAW**				
Description	**Raw Measurement Data**				
Type	Periodic/Polled				
Message Structure	Header	ID	Length	Payload	Checksum
	0xB5 0x62	0x02 0x10	8 + NSV $*$ 24	8 + NSV $*$ 24 Bytes	CK_A CK_B

Table 6.1: UBX RXM-RAW Message Format

In the OASIS system, two types of GPS data are configured to be received by the GPS receiver: NMEA message (Figure 6.2 presents its format) and UBX RXM-RAW message. NMEA message only contains the UTC time information in ASII format. The accurate global time, UTC time, is easy to be parsed from the NMEA message inside a sensor node and this time could be used as the accurate global time source. The other type of data, UBX RXM-RAW message, contains the

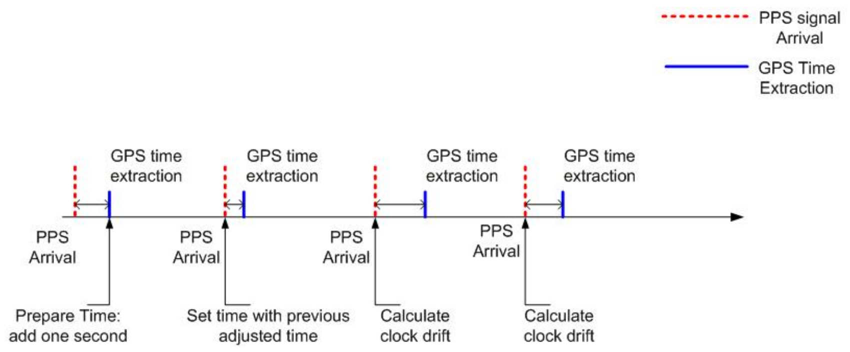


Figure 6.3: GPS Time Synchronization

information of both time and position information (Table 6.1 denotes the format of RAW message). This message will not be processed inside a sensor node but directly packetized and transmitted to the control center. Geologists will use this information to locate the position of each sensor node and measure ground deformations of the volcano environment.

6.3.2 GPS Time Based Synchronization

The primary time synchronization mode is GPS Receiver based synchronization in the OASIS system.

The GPS signal, which contains two types of messages mentioned in Section 6.3.1, is configured to be received every ten seconds through UART and the pulse-per-second (PPS) signal is generated every second (PPS signal is in nanoseconds accuracy, see Section 3.4). The synchronization process could be decomposed into two phases:

- Phase1: GPS data reading and UTC time preparing. At the first time the valid GPS data is recorded, the UTC time information is extracted from the NMEA message and one second adjustment is added to it (UTC time has one millisecond resolution).

- Phase2: PPS signal capturing. When the next PPS signal is captured after preparing the UTC time, the local time value in RTC module is immediately set with previous ready-to-use UTC time. After resetting the local time, the node local time is synchronized with the global time.

The purpose of setting the RTC time value in the next PPS capturing is to minimize the time delay. The GPS data comes to the UART interface after the PPS signal arrived with a short random delay, and the receiving of the whole NMEA message and extraction of UTC time will bring another delay. In order to get a precise global time, setting the global time at the next PPS signal after UTC time ready would remove determinate time delays (see Figure 6.3).

Though GPS data is received every ten seconds, the global time is only set once. After the first time set, the global time will only be adjusted based on the clock drift. The arrival interval of PPS signal is desired to be 1000000 ticks which are assumed to be 1 second, however, due to the clock drift problem (see Section 6.2.2), the arrival interval would decrease and the value could be 999976 ticks in which case a clock tick period is longer than 1 microsecond. The RTC time fire interval will be adjusted periodically based on drifts, which is calculated by the comparison result between assumed PPS arrival time and real local time. For example, if the drift is 27 tick/second, the RTC clock fire interval will be decreased by 1 every $1000000/27$ microseconds. Hence, it compensates the time drift problem and guarantees an accurate global time.

6.3.3 FTSP Based Synchronization

In case of lost of GPS signals, the Flooding Time Synchronization Protocol (FTSP) [34], an existing protocol developed for wireless sensor nodes, is used to achieve an accurate global time. In the original FTSP work [34], timing errors of less than 67 microseconds were reported for an 11-hop network of Mica2 nodes. The protocol worked well in laboratory experiments; however, it experienced significant failures in the field, requiring extensive postprocessing of the data to recover accurate timing for each signal [54]. The original FTSP does not check the validity of synchronization messages, so a node reading an incorrect value for its local clock can corrupt the state of other nodes, throwing off the global time calculation. In the smart sensing implementation, the corrupted packet will be filtered to guarantee only the FTSP packet with correct time is processed to get an accurate global time.

All nodes are running with two mechanisms at the same time: After system started up, both GPS and FTSP time synchronization mechanisms are running, however, if GPS time is valid, the node will not use the time information from FTSP module. The FTSP calculated global time will be used only after GPS is detected to be invalid.

It is possible that several nodes in the network running in GPS based synchronization and the others use FTSP as the time sync modules due to GPS signal losts. The original design was to use static time sync root mode of FTSP. In such a mode, all nodes in the network are forced to synchronize with the multi-hop routing tree root time. However, it is not robust enough in a practical system. First, the sink node may lose GPS signal forever and this will cause the whole network to synchronize with a wrong global time source; and it is possible that the network has multiple sink nodes, which make it impossible to set one node as the time sync root.

Therefore, the dynamic time sync root implementation is selected. However, the original dynamic root mode FTSP is extended: the smart sensing design only allows the node with GPS receiver connected to claim itself as the time sync root after a timeout. With such changes, nodes in the network are not necessary to synchronize with only a fixed node that may have a wrong global time. Finally the time sync root will be the node with the smallest node id and GPS connected. In the worst case, if the whole network lose GPS signals, the node with smallest id will declare itself as the time sync root.

6.3.4 GPS, FTSP auto-switch

We take the advantage of developing the hybrid time synchronization services to ensure the robustness and accuracy. If a node is equipped with a GPS receiver, it will synchronize to the GPS time; if the GPS signal disappears, the system will switch to FTSP mode automatically.

As discussed in Section 1.1.2, the GPS receiver could be damaged in an active volcano environment. If a node loses a valid GPS signal for a threshold time period, that node needs to switch

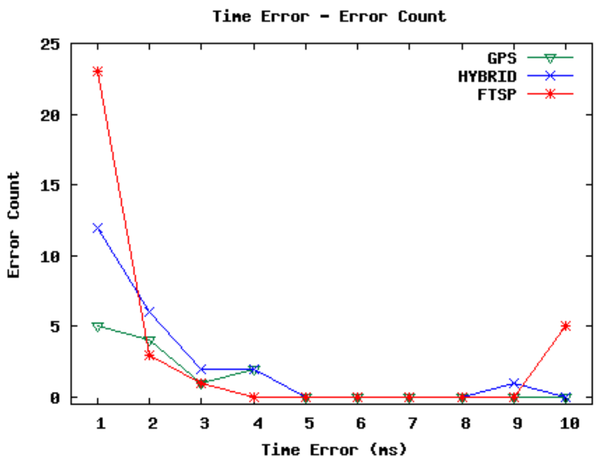


Figure 6.4: Time Synchronization Error

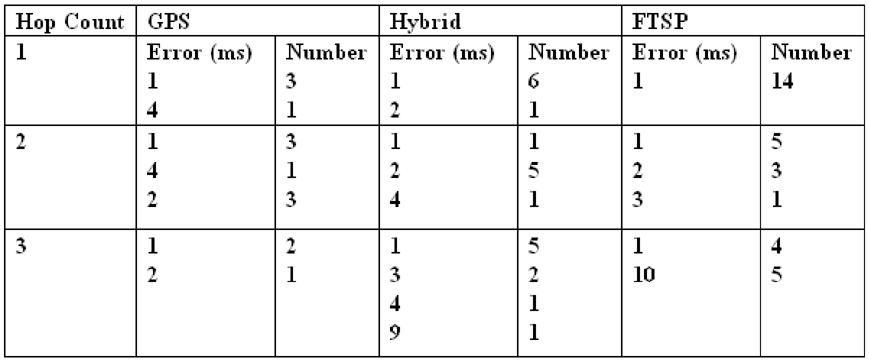
to FTSP synchronization mode in order to maintain an accurate global time. Generally, three situations can drive the synchronization mode from GPS to FTSP: 1) missing PPS signal for over 10 seconds; 2) missing GPS data for over 20 seconds; and 3) receiving two continuous invalid GPS data.

If a GPS signal comes back after a node switched to the FTSP mode, the valid PPS signal and GPS data will change the synchronization mode back to the GPS mode.

6.4 Time Synchronization Evaluation

In the time synchronization evaluation, an irregular sine wave with changing amplitudes and frequencies is supplied to the same analog channel in each node and time-stamps of wave crest and trough on all nodes are recorded. If time values at those two points matches throughout the network, the system is assumed to perform synchronized sampling with accurately time-stamping data. The same program in different time synchronization modes were run for eight hours on a nine nodes network in lab environment.

As shown in Figure 6.4, the time synchronization error of the GPS mode only had five times of one millisecond error in eight-hour experiment and the maximum time error is four milliseconds that only happened twice. In the hybrid mode, there were three nodes connected with GPS and



Hop Count	GPS		Hybrid		FTSP	
1	Error (ms) 1 4	Number 3 1	Error (ms) 1 2	Number 6 1	Error (ms) 1	Number 14
2	1 4 2	3 1 3	1 2 4	1 5 1	1 2 3	5 3 1
3	1 2	2 1	1 3 4 9	5 2 1 1	1 10	4 5

Table 6.2: Hop Count Time Sync Error

the other six were running in the FTSP mode. The one millisecond error increased to twelve times and there was one time of 9 milliseconds error. If the network is running only in the FTSP mode, one GPS is connected to the sink node and the other eight nodes had no GPS connected, the one millisecond error increased a lot compared with the other two modes. However, it is still acceptable that it only happened to 24 data packets among thousands of packets in eight hour running.

Table 6.2 shows the time sync error according to hop count value in the network. Notice that, the single 9 milliseconds error in the hybrid mode and 10 milliseconds error in the FTSP mode all happened to three hop nodes. However, for the reason of the adjustment algorithm running in the node software, this is no accumulative error as system runs.

CHAPTER SEVEN

CONCLUSIONS

7.1 Main Contributions

In this thesis, a smart sensing component design for environment monitoring WSN systems has been proposed and it has been evaluated on the OASIS prototype system. This smart sensing component could help to improve resource utilization and system performance. The main advantages of it are:

- The configurable sensing attribute facilitates sensing component management and increases the system flexibility. It is generic to be integrated into different WSN applications build on different sensor node platforms. It is easy to modify sensing parameters, add or delete sensors and change sensing data processing tasks without reprogramming the whole network.
- It ensures 100% reliable delivery of sink-initiated crucial control commands to targeting nodes in the network and the dissemination could terminate in a short time period with low communication costs.
- The time synchronization design satisfies both the requirements of UTC time based network-wide synchronized sampling and high accurate time-stamp to sensing data. Additionally, the auto-switch mechanism between GPS and Protocol based time synchronization increases the synchronization robustness.
- This smart sensing design provides a framework to perform high frequency multiple channel sensing in a wireless sensor network system. All the developed techniques could be applied to multiple future applications like medical care, urban monitoring, etc.

7.2 Future Work

Though the current smart sensing design has achieved a good result in the running system, it still has improvement potentials:

- Sophisticated event detection algorithms. Though the threshold trigger and STA/LTA based event detection algorithms are easy to implement, they still have disadvantages when the signal noise ratio is low and it is not feasible to detect more complicated event. Therefore, sophisticated and lightweight event detection algorithms need to be developed. Wavelet based event identification mechanism has been used in data mining research and it also has potentials to be applied to wireless sensor networks. Correlated event detection is another highly desired event detection algorithm which could work in an environment with high background noises.

- Linear prediction compression. Though the compression algorithm proposed in the current smart sensing design provides an independent compression characteristic which is not affected by packets loss. However, a more attractive prediction compression should be developed. Such an algorithm is not only independent of packets sequences but also could help to recover data in lost packets which is highly desirable in low link quality networks.

BIBLIOGRAPHY

[1] imote2 datasheet http://www.xbow.com/products/product_pdf_files/wireless_pdf/imote2_datasheet.pdf.

[2] Mda320 datasheet http://www.xbow.com/products/product_pdf_files/wireless_pdf/mda320_datasheet.pdf.

[3] Micaz datasheet http://www.xbow.com/products/product_pdf_files/wireless_pdf/micaz_datasheet.pdf.

[4] Nasa esto website: http://esto.nasa.gov/aistroses.

[5] Oasis: Optimized autonomous space in-situ sensor web. http://sensorweb.vancouver.wsu.edu.

[6] Pressure sensor datasheet http://www.allsensors.com/datasheets/commercial_temp/ds-0091_revb2.pdf.

[7] Daniel J. Abadi, Samuel Madden, and Wolfgang Lindner. Reed: Robust, efficient filtering and event detection in sensor networks. In *The 31st conference in the series of the Very Large Data Bases*, 2005.

[8] Manish Agarwal, Lixin Gao, Joon H. Cho, and Jie Wu. Energy efficient broadcast in wireless ad hoc networks with hitch-hiking. *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, 10(6):897–910, 2004.

[9] K. A. Arisha, M. A. Youssef, and M. F. Younis. Energy-aware tdma-based mac for sensor networks. In *IEEE Workshop on Integrated Management of Power Aware Communications, Computing and NeTworking, IMPACCT 2002, New York*, May 2002.

[10] Sanjit Biswas and Robert Morris. Exor: opportunistic multi-hop routing for wireless networks. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, volume 35, pages 133–144, New York, NY, USA, October 2005. ACM Press.

[11] Steve Chien, Ben Cichy, Ashley Davies, Daniel Tran, Gregg Rabideau, Rebecca Castano, Rob Sherwood, Son Nghiem, Ronald Greely, Thomas Doggett, Victor Baker, James Dohm, Felipe Ip, Dan Mandl, Stuart Frye, Seth Shulman, Stephen Ungar, Thomas Brakke, Lawrence Ong, Jacques Descloitres, Jeremy Jones, Sandy Grosvenor, Rob Wright, Luke Flynn, Andy Harris, Robert Brokenridge, and Sebastian Cacquard. [duplicate] an autonomous earth observing sensorweb. In *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (SUTC'06)*, pages 178–185, Washington, DC, USA, 2006. IEEE Computer Society.

[12] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proc. ACM/IEEE MobiCom*, 2003.

[13] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[14] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, 2002.

[15] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *IEEE ICASSP Conference*, May 2001.

[16] Saurabh Ganeriwal, Ram Kumar, and Mani B. Srivastava. Timing-sync protocol for sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 138–149, New York, NY, USA, 2003. ACM.

[17] Valery Guralnik and Jaideep Srivastava. Event detection from time series data. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 33–42, San Diego, USA, August 1999.

[18] Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. Firewxnet: A multitiered portable wireless system for monitoring weather conditions in wildland fire environments. In *the 4th international conference on Mobile systems, applications and services*, Uppsala, Sweden, June 2006.

[19] Douglas Herbert, Vinaitheerthan Sundaram, Yung-Hsiang Lu, Saurabh Bagchi, and Zhiyuan Li. Adaptive correctness monitoring for wireless sensor networks using hierarchical distributed run-time invariant checking. *ACM Trans. Auton. Adapt. Syst.*, 2(3), September 2007.

[20] Qingfeng Huang, Chenyang Lu, and Gruia-Catalin Roman. Spatiotemporal multicast in sensor networks. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 205–217, New York, NY, USA, 2003. ACM Press.

[21] Jonathan Hui and David Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *The 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys'04)*, 2004.

[22] Texas Instruments. Chipcon cc2420 datasheet http://focus.ti.com/lit/ds/symlink/cc2420.pdf, 2007.

[23] J. B. Johnson, R. C. Aster, and P. R. Kyle. Volcanic eruptions observed with infrasound. *Geophys. Res. Lett.*, 31:14604+, July 2004.

[24] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. John Wiley & Sons, June 2005.

[25] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steve Glaser, and Martin Turon. Wireless sensor networks for structural health monitoring. In *The 4th international conference on Embedded networked sensor systems*, 2006.

[26] P. Kyasanur, R. R. Choudhury, and I. Gupta. Smart gossip: An adaptive gossip-based broadcasting service for sensor networks. In *The Third IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS)*, pages 91–100, 2006.

[27] Philip Levis, Sam Madden, David Gay, Joseph Polastre, Robert Szewczyk, Alec Woo, Eric Brewer, and David Culler. The emergence of networking abstractions and techniques in tinyos. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, page 1, Berkeley, CA, USA, 2004. USENIX Association.

[28] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 2004)*, 2004.

[29] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD '03: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11, New York, NY, USA, 2003. ACM Press.

[30] B. Lo and G. Z. Yang. Key technical challenges and current implementations of body sensor networks. In *Proc. 2nd International Workshop on Body Sensor Networks (BSN 2005)*, April 2005.

[31] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks, 2002.

[32] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The design of an acquisitional query processor for sensor networks. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM Press, 2003.

[33] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless sensor networks for habitat monitoring. In *The First ACM International Workshop on Wireless Sensor Networks and Applications*, 2002.

[34] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49, New York, NY, USA, 2004. ACM.

[35] Ian Marshall, Mark Price, Hai Li, N. Boyd, and S. Boult. Multi-sensor cross correlation for alarm generation in a deployed sensor network. In *Smart Sensing and Context*, volume 4793 of *Lecture Notes in Computer Science*, pages 286–299. Springer Berlin / Heidelberg, 2007.

[36] S. Mcnutt. Seismic monitoring and eruption forecasting of volcanoes: A review of the state of the art and case histories. *Monitoring and Mitigation of Volcano Hazards*, pages 99–146, 1996.

[37] S. R. Mcnutt and C. M. Davis. Lightning associated with the 1992 eruptions of crater peak, mount spurr volcano, alaska. *Journal of Volcanology and Geothermal Research*, 102:45–65, October 2000.

[38] D. L. Mills. Internet time synchronization: The network time protocol. *Global States and Time in Distributed Systems*, 1994.

[39] Moteiv. *Tmote Sky Datasheet http://www.sentilla.com/pdf/eol/tmote-sky-datasheet.pdf*, 2006.

[40] T. Murray and E. Endo. A real-time seismic-amplitude measurement system (rsam). In John W. Ewert and Donald A. Swanson, editors, *Monitoring Volcanoes: Techniques and Strategies Used by the Staff of the Cascades Volcano Observeratory, 1980-1990*, volume 1966 of *USGS Bulletin*, pages 5–10. 1992.

[41] Seung-Jong Park, Ramanuja Vedantham, Raghupathy Sivakumar, and Ian F. Akyildiz. Garuda: Achieving effective reliability for downstream communication in wireless sensor networks. *IEEE Transactions on Mobile Computing*, June 2007.

[42] Yang Peng, Richard Lahusen, Behrooz Shirazi, and Wenzhan Song. Design of smart sensing component for volcano monitoring. In *the 4th IET International Conference on Intelligent Environments*, July 2008.

[43] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min. Z-mac: a hybrid mac for wireless sensor networks. In *The 3rd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2005.

[44] Y. Sankarasubramaniam, O. B. Akan, and I. F. Akyildiz. Esrt:event-to-sink reliable transport in wireless sensor networks. In *The 4th ACM Interational Symposium on Mobile Ad Hoc Networking and Computing(MobiHoc)*, 2003.

[45] Roberto Scarpa and Robert I. Tilling. *Monitoring and Mitigation of Volcano Hazards*. Springer, 1 edition, January 1996.

[46] R. Sherwood and S. Chien. Sensor web technologies: A new paradigm for operations. In *International Symposium on Reducing the Cost of Spacecraft Ground Systems and Operations*, June 2007.

[47] Silicon. 1221j-002 accelerometer datasheet http://www.silicondesigns.com/pdf/1221.pdf.

[48] Siva and B. S. Manoj. *Ad Hoc Wireless Networks Architectures and Protocols*. Communications Engineering and Emerging Technologies. Prentice Hall, 2004.

[49] Fred Stann and John Heidemann. Bard: Bayesian-assisted resource discovery in sensor networks. Technical report, USC/Information Sciences Institute, July 2004.

[50] Fred Stann, John Heidemann, Rajesh Shroff, and Muhammad Z. Murtaza. Rbp: robust broadcast propagation in wireless networks. In *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 85–98, New York, NY, USA, 2006. ACM Press.

[51] U-Blox. Lea-4t datasheet http://www.u-blox.com/products/product_summaries/lea-4t_prod_summary(gps.g4-ms4-05069-b).pdf.

[52] Chieh-Yih Wan, Andrew T. Campbell, and Lakshman Krishnamurthy. Psfq: a reliable transport protocol for wireless sensor networks. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 1–11, New York, NY, USA, 2002. ACM Press.

[53] C. Wang, B. Li, K. Sohraby, and M. Daneshmand. Upstream congestion control in wireless sensor networks through cross-layer optimization. *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS*, 25:786–795, May 2007.

[54] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh. Fidelity and yield in a volcano monitoring sensor network. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 381–396, Berkeley, CA, USA, 2006. USENIX Association.

[55] B. Williams and T. Camp. Comparason of broadcasting techniques for mobile ad hoc networks. In *ACM MobiHoc*, 2002.

[56] Ning Xu, Sumit Rangwala, Krishna K. Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin. A wireless sensor network for structural monitoring. In *ACM Conference on Embedded Networked Sensor Systems*, pages 13–24, November 2004.

[57] Wenwei Xue, Qiong Luo, Lei Chen, and Yunhao Liu. Contour map matching for event

detection in sensor networks. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 145–156, New York, NY, USA, 2006. ACM Press.

[58] Yong Yao and Johannes Gehrke. Query processing in sensor networks. In *First Biennial Conference on Innovative Data Systems Research*, 2003.

[59] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the 21st International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, 2002.

[60] Fenghua Yuan, Wen-Zhan Song, Nina Peterson, Yang Peng, Lei Wang, and Behrooz Shirazi. Lightweight sensor network management system design. In *Fourth IEEE International Workshop on Sensor Networks and Systems for Pervasive Computing*, Hong Kong, March 2008.

[61] Haijiang Zhang, Clifford Thurber, and Charlotte Rowe. Automatic p-wave arrival detection and picking with multiscale wavelet analysis for single-component recordings. *Bulletin of the Seismological Society of America*, 93(5):1904–1912, October 2003.

[62] Rong Zheng. Information dissemination in power-constrained wireless networks. In *INFOCOM, 2006: Proceedings of the 25th IEEE International Conference on Computer Communications*, pages 1–10, 2006.

[63] Michael Zoumboulakis and George Roussos. Escalation: Complex event detection in wireless sensor networks. In *In proceeding of the 2nd European Conference on Smart Sensing and Context*, October 2007.