Design and Implementation of a Mobile Sensor Network Testbed Using SUN SPOTs

Hung-Da Shih, Xiaolan Zhang, David S. L. Wei, Kshirasagar Naik, and Rung Ching Chen

***Abstract*—The rapid development of wireless technologies, such as Bluetooth, IEEE 802.11 and IEEE 802.15.4 protocol has allowed the deployment of wireless sensor networks with thousands of sensor devices in physical environments where the construction of a fixed infrastructure is inconvenient or impossible. System designers have been facing the challenges of using unreliable wireless channels and limited battery power to develop software and protocols for sensor networks. This work focuses on the issues of robustness and efficiency in the design and implementation of wireless sensor networks using Sun SPOTs device. Our implementations not only allow multiple sensor nodes to simultaneously connect to the basestation, but also improve the robustness by handling loss of connection (due to node movements or radio interference). To support flexible and efficient data collection, a sensor collects data during disconnected status, and automatically uploads the stored data when it is in the range of a basestation. The testbed is useful for researches on human mobility pattern, environmental monitoring, and health care, to name a few.**

***Index Terms*—Sensor networks, wireless networks, design and implementation**

I. Introduction

Recent years have seen increasing application of sensor network technology to various domains such as habitat monitoring [16], wildlife monitoring [10], vehicle based sensor network [4], and SensorPlanet Project of Nokia [8]. A plethora of works have studied various aspects of sensor networks, including real deployed sensor networks projects. As illustrated in Fig. 1, a mobile sensor network is made up of base station nodes and sensor nodes. Basestation nodes are attached to host computers (PCs) and serve as gateways between mobile sensor network and Internet. The sensor nodes are either fixed at strategically chosen locations or carried by human beings and/or vehicles. Sensor nodes collect and store sensory data, and upload the data when they come within transmission range of base-station nodes.

There has been a rich body of research works that develop hardware and software architecture for wireless sensor nodes. Among them, the mote series [1], [2], [6], [7] sensor platforms are the most popular platform among research community. Mote family provides different types of sensors with different memory size, processor types, and radio types and so on. However, the number of applications developed is still relatively small, partly due to the lack of adequate tools and languages to facilitate the fast prototyping of applications.

In an effort to develop next generation devices to simplify application development, Sun Microsystems Laboratories developed Sun Small Programmable Object Technology (Sun SPOT). Sun SPOT implements a small Java virtual machine that runs without an operating system. Software developing using Java, a managed runtime language, usually takes less time than using C/C++, a non-managed language, due to the garbage collection, pointer safety, exception handling that comes with Java. Therefore, Sun SPOT has become an ideal choice for carrying out quick experimental and pedagogic projects [18], [17], [13], [9].

This paper presents the design and implementation of a mobile sensor network testbed using Sun SPOTs, as an experimental environment for future research in network routing and sensor data collection schemes. We identify the following design objectives. First, to support mobile sensor node, it is crucial that mobile sensor nodes collect and store data reliably. For this purpose, the disruptive connection to basestation must be handled, and the battery power of the node should be monitored to avoid data loss due to battery depletion. Secondly, we focus on efficiency issue. In order to support high sampling rate, sensor nodes need to collect, store and transmit data efficiently, in terms of battery power, memory space, and processing cycles. The application at the basestation side also needs to handle the incoming sensory data efficiently to support high sampling rate.

The remainder of this paper is structured as follows. In Section II, we present background information about SUN SPOT platform. In Section III, we present the design of the basestation and sensor application. In Section IV, we discuss how we tackle several challenges. An experimental study is presented in Section V. We conclude the paper in Section VI.

II. SUN SPOT System

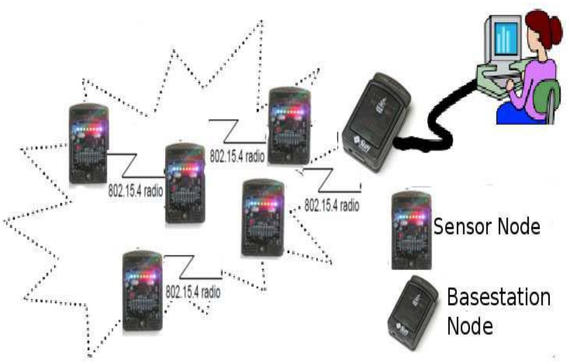


Fig. 1. A mobile sensor network testbed

We provide an overview of the hardware, software and networking protocol stack of Sun SPOT device in this

Manuscript received September 18, 2012; revised October 25, 2012.

The authors are with the Dept. of Computer and Information Science, Fordham University, Bronx, NY, USA (email: hondatp@gmail.com, xzhang@fordham.edu, dsl.wei01@gmail.com, snaik@utaterloo.ca, crching@cyut.edu.tw).

section.

Sun SPOT stands for Sun Small Programmable Object Technology. A detailed description of the SPOT hardware platform is provided in [14]. Here, we describe the main features but leave out technical details.

There are two types of SPOT node configurations: sensor and basestation node. A sensor node (or *eSPOT* node) contains the *eSPOT* main board, a rechargeable battery and the eDemo board. A basestation, on the other hand, only has an eSPOT main board. The basestation node serves as a radio gateway between SPOTS sensor nodes (and other 802.15.4 devices) and the host workstation, and is powered by a USB connection to a host workstation.

The *eSPOT* main board is a common component for basestation and sensor configurations. It contains an ARM920T core processor, a 4MByte Flash memory and a 512 KB random access memory (RAM), power management circuit, 802.15.4 radio transceiver and antenna, battery connector, and daughterboard connector. The memory contents of RAM are maintained as long as there is power supply, even during periods of off or "deep-sleep". The main board contains a timer chip, a USB interface and two LEDs. The timer provides timing service used for implementing periodic tasks.

The sensor node comes with a battery board with a rechargeable battery which is recharged whenever the USB interface is connected to a PC or powered USB hub.

Like the battery board, the *eDemo* board (i.e., sensor board) is only available on sensor nodes. It contains a processor, a flash memory, a light sensor, a temperature sensor, an accelerometer, eight tri-color LEDs and two pushbuttons. The LEDs and pushbuttons provide an interface through which end users interact with the sensor nodes.

Sun SPOT is especially designed to facilitate implementation and deployment of new sensor applications [15]. It implements a Java Mobile Edition implementation, Squawk, supporting Connected Limited Device Configuration (CLDC) 1.1 and Mobile Information Device Profile (MIDP 1.0) which are key elements of J2ME. Squawk also implements operating system functionalities such as thread scheduling, interrupt handling and device driver support.

Sun SPOT network stack follows the commonly used five-layered structure. The physical and MAC layer partially implements IEEE 802.15.4 [5], [12] standard. The networking layer implements the AODV [3] routing algorithm. Above MAC and networking layer, SPOT supports two communication services. The *radiostream* service provides TCP-like reliable, buffered, stream-based communication between two devices. The *radiogram* service provides UDP-like datagram-based communication between two devices. The radiogram protocol provides no guarantee about delivery or in-order delivery. Datagrams sent over more than one hop could be lost, delivered more than once, and delivered out of sequence. Due to the MAC layer acknowledgement and retransmission scheme, radiograms sent over a single hop do not experience above problems, but might be delivered more than once.

SPOT also supports broadcast communication through broadcast datagrams, which is also an unreliable communication.

III. Building a Mobile Sensor Network

In this section, we first present the design objectives in Section III.A, and then present the system architecture in Section III.B. We illustrate the communication between sensor nodes and host application in Section III.C, and outline the main system components of the sensor and basestation applications in Section III.D.

A. Design Objectives

We identify our main design objectives as follows. First of all, the mobile sensor network needs to run continuously in various scenarios. It should support multiple simultaneous mobile sensor nodes, with each sensor node moving in and out of the range of basestation during the data collection process. Second, the limited memory space and battery power of the sensor node needs to be used efficiently to support higher sampling rate or increase network lifetime.

B. System Overview

We base our system on a telemetry demo program in SPOT SDK, which includes two applications, the basestation application and the client application that respectively run on the basestation node and sensor nodes as illustrated in Fig. 1. The basestation node is connected to a PC running Windows via an USB connection, and runs in a dedicated mode, meaning that it runs within the same Java virtual machine as the host application.

The basestation application accepts data uploaded from sensor nodes, and provides GUI (as shown in Fig.2) through which the end user views the sensory data, issues commands to sensor nodes. The sensors panel on the left of the window displays the list of sensor nodes (the MAC addresses) that are connected to the basestation, using different colors to indicate their status. In the bottom, there is a control panel for the end user to issue commands to sensory nodes, e.g., to calibrate and configure the sensors, to adjust sensor sampling rate, and to start or pause data collection. For example, the "Battery" button for checking the battery status of the selected sensor. The data collected from the selected sensor is plotted in the main panel of the window in real time. Whenever a sensor node is disconnected (due to its moving out of transmission range or poor link quality), the basestation saves sensory data collected from the sensor node to a file.

The sensor application runs on a sensor node. Upon startup, it broadcasts a connection request message and listens for basestation broadcast announcement. When a basestation is discovered, it connects to the basestation, i.e., builds a unicast connection with the basestation, and subsequently transmits sensory data to basestation. When it is disconnected from a basestation, it samples and stores sensory data to its local memory. The sensor application also monitors the battery level, and saves the sensory data and shuts down if the battery level is below a certain threshold. When in disconnected state, end user can use push button to start or to pause data collection. When the sensor connects to basestation, sensory data stored locally will be uploaded to the basestation. Fig 3 illustrates the state diagram of a sensor.

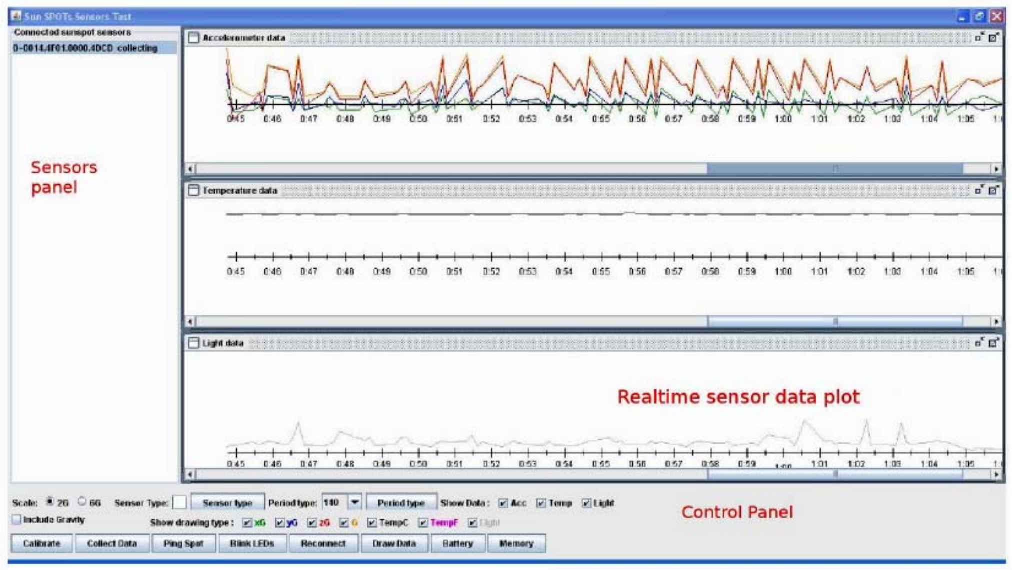


Fig. 2. Basestation application GUI

C. *Communication between Basestation and Sensor*

As we currently focus on one-hop communications between one base-station and several sensor nodes, we use radiogram and turn off routing mechanism on all devices. Similarly, broadcast is configured to be transmitted over one hop only. We first describe how basestation and sensor node discover each other and set up a connection, and then describe their communication in connected state. We follow the communication mechanism of the telemetry demo, with some modifications to detect and handle lost connections.

During device discovery, both sensor and basestation broadcast and listen for messages on the same broadcast channel. On receiving a broadcast message from the other device, the MAC address of the other device can be obtained and subsequently they can set up unicast radiogram connection. More specifically, after the sensor application starts up, it periodically sends out a broadcast message on port 42. When the basestation receives the broadcast message, it sends a broadcast message on the same port, sets up a unicast radiogram connection with the sensor on port 43, and starts to listen on the connection. When the sensor receives the reply from the basestation, it sets up a unicast radiogram connection with the basestation on port 43, and starts to listen on this connection for command messages from the host application. Once both sides set up the radiogram connection with each other, the sensor is said to be “connected” to the basestation.

While in the connected states, the sensor listens on the radiogram connection for command messages from the basestation and replies with a message with command or sensor data in it. The basestation listens on the connection for data packets, and when the end user clicks on a control button in the GUI, it sends the corresponding command message to the sensor.

As both communication mechanisms used (unicast datagram and broadcast) provides no acknowledgement, there needs another way for the sensor to monitor the status of the basestation and the connection. For this purpose, every 10 seconds, the basestation sends out a broadcast message on port 43 as a “heartbeat” message. The sensor in connected state periodically listens for this heartbeat message on broadcast port 43. If the sensor does not receive the message for a certain amount of time (12 seconds), it infers that it is out of range of the basestation, or the server application has exited, and closes the unicast connection (and enter “disconnected” state), and starts LocateService to rediscover the basestation.

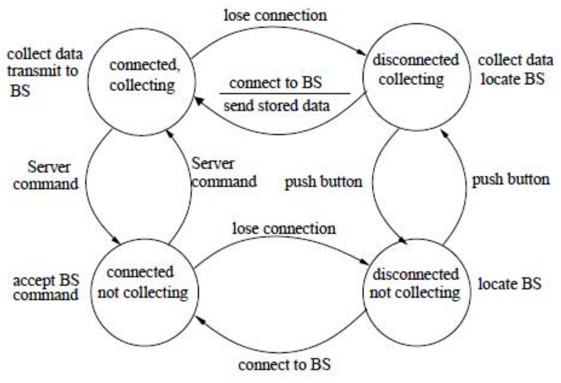


Fig. 3. Sensor node state diagram

D. *System Components*

In this section, we describe the main system components (i.e. Java classes), and the interaction among them.

The main components of the server application are SensorFrame, ListeningSpots, AccelerometerListener, SensorData and GraphView. On startup, SensorFrame is started, which initiates ListeningSpots and GraphView. ListeningSpots listens for broadcast messages from sensors, broadcast its information periodically, while GraphView plots sensory data in GUI window. For each connected sensor, an AccelerometerListener and SensorData component are created to communicate with the sensor and to save received sensory data respectively.

The sensor application consists of SensorMain, LocateService, PacketReceiver, PacketTransmitter, AccelMonitor, and flashMem components. Upon starting up, SensorMain is started which initiates LocateService which sends and receives broadcast messages to discover nearby basestation nodes. When it connects to a basestation node, it starts PacketReceiver and PacketTransmitter to receive and transmit message.

When the data collection starts, AccelMonitor is started to collect sensory data periodically. In disconnected state, AccelMonitor passes data to flashMem which stores data in the Flash memory. In connected state, AccelMonitor stores the data in RAM until the amount of data exceeds a certain threshold, and then passes the data to PacketTransmitter to transmit to the basestation.

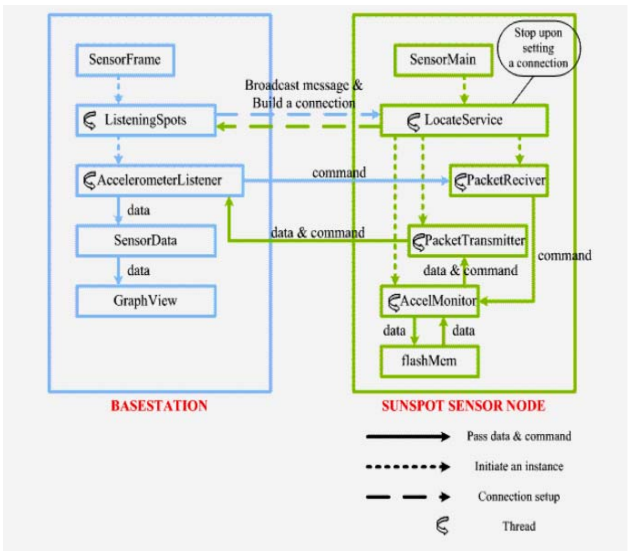


Fig. 4. System components

IV. Achieving Robustness, Flexibility and Efficiency

A. Robust Connection Maintenance

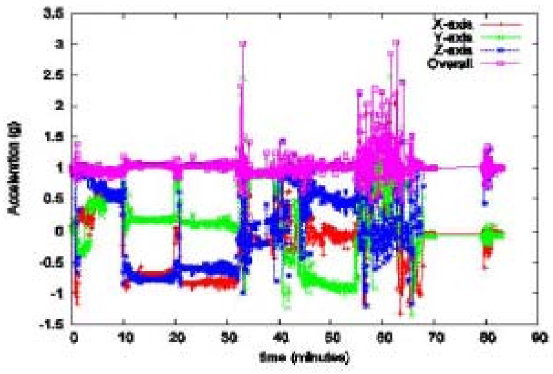
To handle multiple sensor nodes, the server keeps track of the status of its connection with each sensor. When a sensor disconnects from the server, the server closes the (radiogram) connection and stops related functions. Our experience suggests that a proper closing up is necessary for releasing up lower layer networking resource. Whenever the sensor loses connection with the basestation, it starts LocateService to rediscover the basestation.

B. Flexible Sensor Data Collection

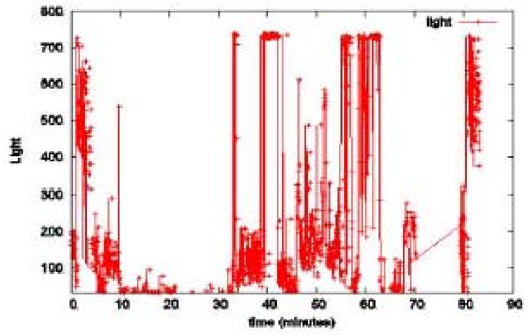
We allow sensor to collect sensory data and store them in local memory when it is not connected. Due to the small size (512KB) and volatility of RAM, we use FLASH memory to save sensory data collected during period of disconnection. Unlike RAM, FLASH is permanent memory: even if the battery is depleted, the content of FLASH is preserved.

The size of Flash memory is 4MB. Libraries and applications have taken a substantial amount of space, leaving approximately 524226 bytes for saving sensory data. Sun SPOT provides two ways to access Flash memory, Record Store Manager (RMS) and FlashFile. As FlashFile is a low-level mechanism, we choose the former for it is relatively simpler. The RMS manages the Flash memory in the unit of record, and uses record store to manage a set of records.

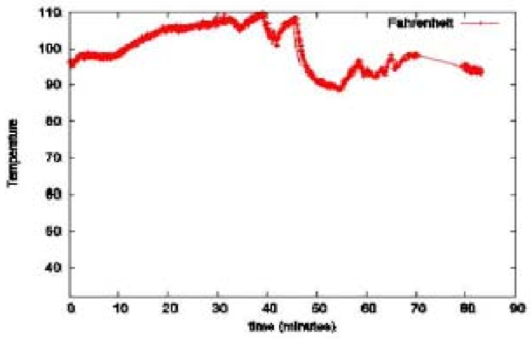
Using different record size (i.e., how much data to store in each record) affects the total amount of accessible Flash memory. If we store only one sample (32 bytes) in a record, the maximum amount of data that can be stored is 40480 bytes, due to the overhead in managing the Flash File Allocation Table. Currently, we store 6 samples in each record, which allows us to access up to 60480 bytes Flash memory. When the sensor application uses up the flash memory space, it deletes the oldest 3 records.



(a) Accelerometer sensory data



(b) Light sensory data



(c) Temperature sensory data

Fig. 5. Sensor data collected in experimental studies

C. Efficient Data Collection and Uploading

For different applications, the required sampling rate varies greatly. For example, if accelerometer sensory data are used for measuring the SPOT's orientation or gesture recognition, a sample rate of 10-20 readings per second will suffice. Considering the various hardware and software components, the maximum sample rate is 320Hz [11], i.e., a reading every 3.125 milliseconds, a rate that can be sustained and still allow for other computation to be done.

When running in connected state, the sensor stores data in RAM until a certain amount of data has been collected. It then packs multiple readings into one packet so that the communication overhead (of sending packet headers and lower level protocol overhead) is amortized over larger data payload. Each sensory reading, including accelerometer, light and temperature sensor reading, is 32 bytes long. Currently, the sensor waits until 6 samples have been collected. Then, it packetizes the data into 3 packets and transmits them to the basestation. Each data packet carries 12 byte sensor data header including the packet sequence number (1 byte), timestamp (8 byte), type of sensor data (1 byte), and the number of samples in the packet (1 byte).

D. Transmission Rate Control

When a sensor first connects to basestation, it needs to uploaddata that have been gathered to the basestation. A naive solution of transiting all stored data in one batch leads to failure of the sensor application. First, the radiogram connection provides no rate control, and when a large number of packets are sent in a short duration of time, various components of networking stack experience failure such as buffer overflows. Second, if AccelMonitor thread transmits all stored data out in one batch, it prevents other threads such as PacketReceiver or some system threads from being scheduled, causing various problems.

To solve this problem, we introduce simple rate control to the sensor application: AccelMonitor sends 9 packets in each round, i.e., each time it wakes up. As each time AccelMonitor wakes up, only one new sensory sample is collected while 9 packets are sent, eventually all sensory data are uploaded. We currently perform data uploading from AccelMonitor thread, whose period is determined by the sampling rate. Using a separate thread to upload data collected disconnection allows more flexibility in data uploading rate, but leads to the overhead of having one more thread.

V. Experimental Result

We perform several experiments where the author carried a sensor node while commuting from his home to his office, during which sensory data were collected with a sampling rate of one reading per 2 second. Fig. 5 plots the accelerometer, light and temperature data collected. We interpret the collected data by relating it to the actual scenario under which the data was collected.

During minute 0 to 10, the author left his house, walked to the subway station. The accelerometer data collected show a certain level of variations. From minutes 10 to 33, the author sat in a subway train, with the sensor node in his backpack. The values of accelerometer data vary little while the values of light sensory data stay near zero. The temperature readings gradually increased, as it measures the temperature of the chip which increases when the sensor node is on. Around minute 33, the author got off the train and walked to another platform, and transferred to another train at around minute 35. Accordingly, the accelerometer data show more variations. From minutes 35 to 57, he sat in the train, and took out the sensor node from the backpack from minutes 39 to 57. He held and swung the sensor from minutes 39 to 45, and we observe a larger variation in the accelerometer readings. At around minute 45, he put down the sensor on next seat. As a result, the accelerometer readings show a smaller variation, while the temperature readings decreased gradually as the sensor cooled down. At around minute 57, the author got off the train and started walking to his office, holding the device in hand, leading to a larger variation in the accelerometer data. Upon arriving at the office, the sensor node was turned off and restarted a few minutes later to upload data to the basestation.

VI. Conclusions

We designed and implemented a mobile sensor network testbed using Sun SPOTs, focusing on the issues of lightrobustness and efficiency. Our implementations support multiple sensor nodes simultaneously connecting to the basestation, and handle loss of connection and performing reconnection. To support flexible and efficient data collection, the sensor program collects sensory data during disconnected status, and automatically uploads the stored data when it is in the range of a basestation. When the sensor node disconnects from the basestation, the host application saves data uploaded from the sensor into a file. Possible future works include extension to fully support different routing schemes and sensor data aggregation schemes, and study of energy efficiency issue.

References

[1] Crossbow Technology. [Online]. Available: http://www.xbow.com/Products/wproductsoverview.aspx.

[2] Intel imote. [Online]. Available: http://www.ee.ucla.edu/mbs/ipsn05/demo/11_RKling.pdf.

[3] C. Perkins, E. B. Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," *IETF. RFC*, 2003.

[4] B. Hull, "CarTel: A Distributed Mobile Sensor Computing System," in *SenSys*, 2006.

[5] J. A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile, "A Developing Standard for Low-Power Low-Cost Wireless Personal Area Networks," in *IEEE Network*, 2001, vol. 15, no. 5, pp. 12-19.

[6] J. Hill, "System architecture for wireless sensor networks," *University of California at Berkeley*, 2003.

[7] J. Hill, "System architecture directions for networked sensors," in *Proc. of ASPLOS-IX Conference*.

[8] Nokia Sensor Planet project. [Online]. Available: http://www.sensorplanet.org.

[9] Y. Iwasaki, K. Naito, K. Mori, and H. Kobayashi, "Implementation of Energy Saving Mechanisms for Sensor Networks with SunSPOT Devices," in *ICMU*, 2012.

[10] P. Juang, M. M. L. S. P. H. Oki, Y. Wang, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with Zebranet," in *Proc. of ASPLOS-X Conference*, 2002.

[11] R. Goldman. Using the lis3l02aq accelerometer. A sun spot application. [Online]. Available: http://www.sunspotworld.com/docs/AppNotes/AccelerometerAppNote.pd

[12] S. C. Ergen. Zigbee/IEEE 802.15.4 summary. a preprint. [Online]. Available: http://www.sinemergen.com/zigbee.pdf.

[13] Corona. A distributed query processor for sunspot wireless sensor network platform. [Online]. Available: http://sydney.edu.au/engineering/it/wsn/corona/?corona.

[14] Sun Labs. Sun SPOT Theory of Operation. [Online]. Available: http://www.sunspotworld.com/docs/Red/SunSPOTTheoryOfOperation.pdf.

[15] Sun SPOT Developer's Guide. [Online]. Available: http://www.sunspotworld.com/docs/Yellow/SunSPOTProgrammers-Manual.pdf.

[16] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler, "Lessons from a sensor network expedition," *EWSN*, 2004.

[17] D. Tyman, N. Bulusu, and J. Mache, "An Activity-Based Sensor Networks Course for Undergraduates with Sun SPOT Devices," *SIGCSE*, 2009.

[18] D. V. D. Akker, B. Braem, and C. Blondia, "adapting the Sun SPOT architecture for MAC development," *Annual Symposium of the IEEE/CVT*, 2009.

Hung-Da Shih received the Bachelor of Business Administration in the Department of Computer and Information Science from the Soochow University in 2004. From September 2007 to present, he has been with the Department of Computer and Information Sciences, Fordham University, as a graduate student pursuing his Master degree in Computer Science.

Xiaolan Zhang holds a Ph.D. in Computer Science from University of Massachusetts Amherst. She received her B.Sc. degree from the Beijing University in China, and her M.Sc. degree from University of Massachusetts Amherst. She is currently an assistant professor in the Computer and Information Science Department at Fordham University in New York, NY, USA. Her research interests include mobile wireless networks, mobility modeling and Disruption Tolerant Networks.

David S. L. Wei received his Ph.D. degree in Computer and Information Science from the University of Pennsylvania in 1991. Dr. Wei has authored and co-authored more than 80 technical papers in the areas of distributed and parallel processing, wireless networks and mobile computing, optical networks, peer-to-peer communications, and cognitive radio networks in various archival journals and conference proceedings. He is a lead guest editor of IEEE Journal on Selected Areas in Communications (J-SAC) for the special issue on Networking Challenges in Cloud Computing Systems and Applications. He was a lead guest editor of IEEE J-SAC for the special issue on Mobile Computing and Networking, and was a guest editor of IEEE J-SAC for the special issue on Peer-to-Peer Communications and Applications. Currently, Dr. Wei focuses his research efforts on wireless sensor networks, cognitive radio networks, cloud computing, and intelligent transportation system.

Kashirasagar Naik is an associate professor in the Dept. of Electrial and Computer Engineering at the University of Waterloo, Ontario, Canada. Previously, he was a software development engineer for Wipro Technologies in Bangalore, India. Dr. Naik has contributed to numerous journals and conference publications in the area of software testing.

Rung-Ching Chen received the B.S. degree from department of electrical engineering in 1987, and the M. S. degree from the institute of computer engineering in 1990, both from National Taiwan University of Science and Technology, Taipei, Taiwan. In 1998, he received the Ph.D. degree from the department of applied mathematics in computer science sessions, National Chung Tsing University. He is now a professor at the Department of Information Management and a Dean at college informatics in Chaoyang University of Technology, Taichung, Taiwan. He has been a Fellow of IET since July, 2011. His research interests include web technology, domain ontology, pattern recognition and knowledge engineering.