Air Programming on Sunspot with use of Wireless Networks

Vimal Adodariya[1] Dhaval Shingala[2] Darshak Sojitra[3] Rajnik Vaishnav[4] Nirav Kansagra[5]

[1]HOD(CE) [2]Vice Principal [3, 4, 5]Sr. Lecturer

[1, 2, 3, 4, 5] Aarsh Mahavidhyalaya Diploma Engineering College, Rajkot-Bhavnagar Highway-360020 (Gujarat)

Abstract— Wireless Sensor Networks (WSN) provides us an effective means to monitor physical environments. The computing nodes in a WSN are resource constrained devices whose resources need to be used sparingly. The main requirement of a WSN is to operate unattended in remote locations for extended periods of time. Physical conditions, environmental conditions, upgrades, user preferences, and errors within the code can all contribute to the need to modify currently running applications. Therefore, reprogramming of sensor nodes is required. One of the important terminologies that are associated with WSN network is that of OVER THE AIR PROGRAMMING. This concept has been utilized so far as for Imote2 sensors that has been relatively utilized for the processing of the Deluge port (DP). They have so far been able to successfully reboot each application. But this rebooting is still not reliable and secure as there are certain security features that are affected in the processing. I will provide a more reliable, robust and secure system that would have enriched functionalities of that of OTA programming on SUNSPOT. Some important features of the SUN SPOT that I will be utilizing in this practical approach is that it supports isolated application models such as sensor networks, it allows running multiple applications in one. It does not create overhead on the entire system node as it provides lower level asynchronous for proper message delivery. It also supports migration from one device to another. This paper focuses on developing a multi-hop routing protocol for communication among Sun SPOT sensor nodes and a user front-end (i.e. visualizer) to visualise the collected values from all the nodes. To test the routing protocol before deploying it to sensor nodes, a simulation using J-Sim is created.

I. INTRODUCTION

Over-the-air programming (OTA) refers to various methods of distributing new software updates or configuration settings to devices like cell phones and set-top boxes. In the mobile world these include over-the-air service provisioning (OTASP) over-the-air provisioning (OTAP) or over-the-air parameter administration (OTAPA), or provisioning handsets with the necessary settings with which to access services such as WAP or MMS. Some phones with this capability are labelled as being "OTA capable. "As mobile phones accumulate new applications and become more advanced, OTA configuration has become increasingly important as new updates and services come on stream. OTA via SMS optimizes the configuration data updates in SIM cards and handsets and enables the distribution of new software updates to mobile phones or provisioning handsets with the necessary settings with which to access services such as WAP or MMS. OTA messaging provides remote control of mobile phones for service and subscription activation, personalization and programming of a new service for mobile operators and Telco third parties. When OTA is used to update a phone's operating firmware, it is sometimes called Firmware Over The Air (FOTA). For service settings, the technology is often known as Device Configuration.

Wireless Sensor Network (WSN)

A Wireless Sensor Network (WSN) is composed of small and highly resource-constrained sensor nodes that monitor some measurable phenomenon in the environment, e.g., light, humidity, or temperature. WSNs are deployed in a steadily growing plethora of application areas. These range from military (e.g., security perimeter surveillance) over civilian (e.g., disaster area monitoring) to industrial (e.g., industrial process control). Application scenarios of WSNs typically involve monitoring or surveillance of animals or humans, infrastructure, or territories. Their long-life and large-scale design, various deployment fields, and changing environments necessitate the feasibility of remote maintenance and in-situ reprogramming of sensor nodes using a so-called Over-The- Air Programming (OTAP) protocol. In particular, if sensor nodes are inaccessible after deployment, a reliable OTAP is crucial. I believe that in a plurality of WSNs, the network-wide dissemination of program code is not appropriate. Within a single WSN, the heterogeneity of sensor hardware, the deployment of manifold sensor technologies, the diversity of sensing and communication tasks, and possibly the event and location dependency of software require a flexible, group-wise selective OTAP approach in order to be able to efficiently reprogram a subset of nodes. Furthermore, securing the OTAP protocol is imperative in order to protect the OTAP from unauthorized reprogramming attempts, i.e., to prevent reprogram node attacks.

I believe that running Java technology on a wireless sensor device will simplify application and device driver prototyping, thereby increasing the number of developers able to build applications, as well as their productivity; resulting in more interesting applications sooner. The Java platform brings with it garbage collection, pointer safety, exception handling, and a mature thread library with facilities for thread sleep, yield, and synchronization. Standard Java development and debugging tools can be used to write wireless sensor applications. Further, I provide tools for managing, deploying and monitoring these devices in a graphical user interface called SPOT World. This paper concentrates on the ways in which development of sensor applications in such an environment is simplified.

What are Java Sun SPOTs?

Sun SPOTS are wireless sensor networks that run MIDP-based applications (Mildest) on a "Squawk VM" (a small Java ME virtual machine), which provides the ability to run these wireless transducer applications "on the metal". These Mildest have access to the sensor capabilities of the Sun SPOTS and can migrate (along with state information) from one Sun SPOT to another.

II. RELATED WORK

A. *Commercial Sensor Nodes*

Wireless Sensor Network has emerged as a very hot topic in the networking area recently due to its widespread applications in industry, military, environment as well as real life. There are many commercial Wireless Sensor nodes available nowadays, and some of the most popular ones such as: TMote Sky, MICA Mote, Intel Mote, or Sun SPOT. The first three are described below.

1) TMote Sky:

TMote Sky is one of the most popular Wireless Sensor Node today. It is the next-generation mote platform for extremely low power, high data rate sensor network applications. It has integrated sensors, radio, antenna, microcontroller and programming capabilities. With 1MB Memory, 10 KB RAM, 48KB Memory Flash and 250 kbps Radio bandwidth, TMote Sky can offer a robust solution for many wireless sensor applications in industry, environment and so on. TMote Sky supports TinyOS as the operating system for each sensor node and applications can be wirelessly programmed to the TMote Sky module.

2) MICA Mote:

MICA is a second generation mote module used for research and development of lowpower, wireless sensor networks. It was developed by UC Berkeley's research group on wireless sensors. MICA Mote has 4KB RAM, 4 Mb Memory Flash and the Radio bandwidth of 40kbps. In comparison with TMote Sky, MICA Mote is less powerful and its memory has less capacity. MICA Mote also uses TinyOS as the operating system.

3) Intel Mote:

Intel Mote hardware is a modular, stackable design that includes 64MB RAM, 512KB Flash and a very high Radio bandwidth of 600kbps. Like TMote Sky and MICA Mote, Intel Mote software is based on TinyOS and in the future it will be supposed to incorporate security features such as authentication and encryption.

	MICA Mote	Intel Mote	TMote Sky	Sun SPOT
Memory	128 KB	N/A	1 MB	N/A
Ram	4 KB	64 KB	10 KB	512 KB
Flash	4 Mbit	512 KB	48 KB	4 MB
OS supported	TinyOS	TinyOS	TinyOS	Java VM (no OS)

Table. 1: Compare sensor nodes' hardware

As we can see from the table, Sun SPOT has much larger Ram and Flash memory capacity than MICA Mote, Intel Mote or TMote Sky has. With the capacity of 512MB, Sun SPOT's RAM is 8 times more powerful than TMote Sky's RAM (64KB); meanwhile, Sun SPOT's Flash Memory (4MB) is 8 times larger than Intel Mote's Flash Memory (512K). In addition, Sun SPOT is based on a 32-bit RAM CPU and 11 channels 2.4GHz, IEEE 8.15.4 radio chip which are among the latest Wireless Sensor Network technologies. It is obvious that Sun SPOT nodes are the most powerful commercial WSN nodes at this moment. With large memory on-device, high speed CPU and many other advantages, Sun SPOT has a great ability to perform complex process and control operations; it also radically simplifies the process of developing wireless sensor and transducer applications. Furthermore, Sun SPOT is very promising in solving several technical challenges in the Wireless Sensor Network area such as:

- Current development tools for creating and investigating wireless sensor are difficult to use and unproductive.
- Within tight resource constraints of sensor nodes and affordable cost, effective security mechanisms are really difficult to be implemented.
- Current sensor nodes have limited processing capability, which restricts signal analysis and control processes.
- Unique characteristics of these new small devices present challenges for networking. New protocols and standards must be created for sensor nodes to communicate with each other.

III. DESIGN AND IMPLEMENTATION

The main tools that were used during the project included Java and its libraries, Squawk SDK and J-Sim. The hardware involved in the project included Sun SPOT nodes and a basic PC with the packages mentioned above installed. Since my project requires my code to be written in Java to run on the Sun SPOT nodes, I have chosen java and its related packages to develop the entire system. For the user interface an open source java graphing library (J2D) was used to present line graphs of the temperature and light recorded for selected nodes. This library was used for two key reasons, an external graphing library was needed since java's library's doesn't provide an easy way to develop graphs, and it did exactly what I needed, while still being simple to learn and use.

The Sun SPOTs SDK was essential to the project as I needed to use its libraries to develop code for the Sun SPOT nodes. By reading simple examples found with the SDK resource i will able to develop the needed code for the system. J-Sim, a simulation tool was used for the first task of the project which was to simulate the path taken from each node to the sink and ensure it was the shortest. J-Sim was chosen over other possible candidates such as bSpots for a number of reasons such as ease of use, ability to do what is required and familiarity with the tool.

The use of Sun SPOT Nodes was essential for the successful completion of the project. As mentioned in section one of this document, the code that I developed in the earlier stages of the project needed to be migrated to the Squawk environment and then run on Sun SPOT nodes, hence showing the necessity of the nodes.

Coding and Design Methods

A small concern that played a part during coding was to ensure that each class I developed had low dependency/coupling between other classes. Initially some of the user interface classes showed to have high coupling as

many of the classes depended heavily on the Visualiser GUI class. Quickly fixed by dividing the Visualier GUI class into many classes. I chose to ensure my code had low coupling because this is a key sign of a well-structured computer system and makes code a lot easier to understand and maintain. Due to the nature of the project I felt there were no common design patterns that I could make use of, except those forced upon us, i.e. Java's swing library follows a structured way to add components to a window.

To improve the look and readability of code in general, I decided to implement a standard coding strategy, on how I code should be written, in terms of spacing between braces etc.

Algorithms

In this project, I need an algorithm to build the multi-hop routing protocol. Among several available routing algorithms I chose Bellman-Ford, a distance-vector algorithm, to find the shortest path from each sensor node to the sink. The routing algorithm should be simple to avoid overhead since each sensor node has only a limited memory capacity. Using Bellman-Ford, each node estimates distance to every other node in the network (distance vectors) and transmits the information to its neighbours, then it receives similar distance-vectors from its neighbors. Bellman-Ford algorithm can adapt to traffic changes and link failures as well as is suitable for networks with multiple administrative entities. Hence, Bellman-Ford was a good choice for building a multi-hop routing protocol in my project.

IV. SYSTEM ARCHITECTURE

There are four main layers to my application as a whole, the database layer, visualiser (GUI Layer), J-Sim(Simulation Layer) and Sensor nodes. Each layer can function separately, but only when placed together will the complete application work as desired.

A. The GUI Layer:

It is used to present the user with statistical data that has been retrieved by each node. This includes the name, id, status, temperature, light and location of each node. All this data is presented in tables, but only the most current temperature and light data for each node is shown. To show how temperature and light performs over time, the visualiser also presents the user with two line graphs. The temperature graph clearly indicates how the temperature of selected nodes varies over time. Similarly the Light Line graph performs the same function with light over time.

B. The Database Layer:

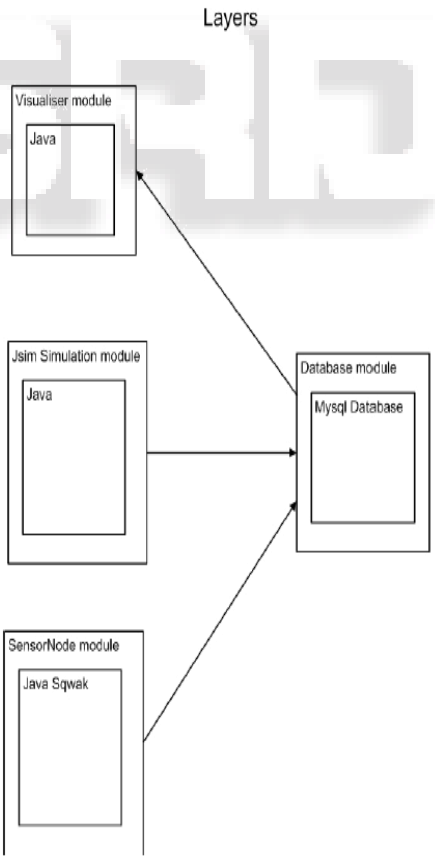
It essentially allows other layers to store and retrieve data from the database. The database used is MySQL, and it was chosen due to its light weight nature and it being open source. There are 3 tables in the database, Node, Data Node and Neighbor Node. As each name suggests, Node stores all data referring to each node such as each the node's id, name, location, power and data rate. The Data Node table keeps the statistical data gathered by each node such as the temperature, light, status and the time the data was gathered. The Neighbor Node table is used to keep track of each nodes closest neighbor, so it simply stores two node ids, one of the node and the other of its neighbor.

C. The Simulation Layer

It is built using J-Sim is used to simulate a many-to-one data collection system of wireless sensor networks. The main purpose of developing the simulation is to test the simple many-to-one routing protocol before migrating all the codes to J2ME/Squawk environment. To support simulating Wireless Sensor Networks, J-Sim provides object-oriented definitions of (i) target, sensor and sink nodes,(ii) sensor and wireless communication channels, mobility model and power model. Input data such as network's configuration that includes network's size, locations of all the nodes in the area and the range of each sensor node is written in the form of a Tcl/Java file. The output of the simulation will be stored into the database which is connected to the visualiser so that the simulation's results could be visualised.

D. The Sensor Nodes Layer

It involves working with Sun SPOT sensor nodes. There are two main types of hardware that are involved in this layer: the base station and target sensor node. Each type of hardware requires a specific type of code to run. The base station's code allows it to receive data from target nodes and to store that data into the database. The target sensor node's code allows it to read data(temperature/light) from the sensor node through an interface (provided in Sun SPOT SDK) and send that data to the base station.



Relationship between J-Sim, Database and Visualiser modules

Fig. 1: Diagram showing Relationships between the Main Modules

The following sequence diagrams and the layers diagram seen above, clearly show the interaction between components/modules in my application. The first diagram shows how the simulation and the visualise work together to report the necessary data to the user. As you can see the J-Sim module, goes through each node, generates random temperature and Light data, sends it to the sink (via the shortest path through other nodes in the network) and then stores it in the database. This is repeated every 10 seconds until the simulation is stopped by a user. At the same time the visualiser is running and it gathers temperature, light and other data from the database for each node and displays it to the user. This is also repeated every 10seconds.

V. MODULES AND THEIR INTERFACES

A. Visualiser Modules

See class diagram below.

.

Visualiser's class diagram

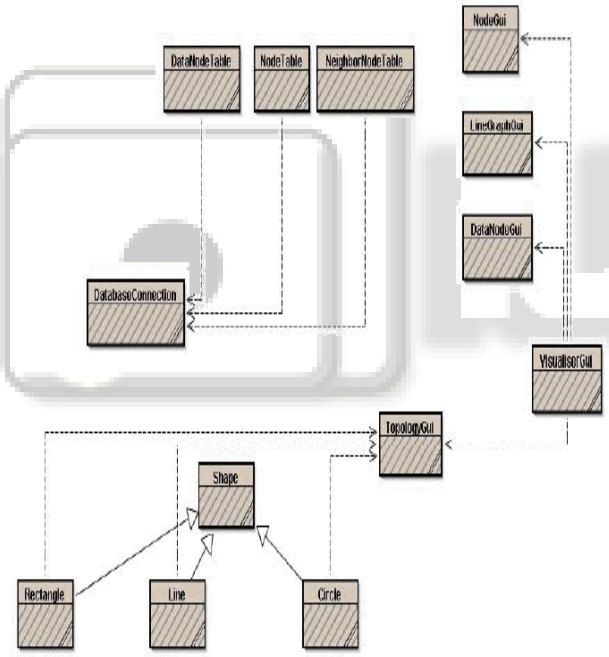


Fig. 2: Class diagram of Visualiser tools

1) DataNodeGui:

This module handles all the components that are placed in the data tab. The main method used to gather the information required from the database is get Table Data ,and is called every time a user clicks the refresh button to update the details.

2) LineGraphGui:

This module handles all the components that are placed in the Temp Line graph and Light Line graph tabs. Add New Trace is the key method used, as it is called every time a user selects a checkbox next to a node in the node table

3) NodeGui:

This module handles all the components that are placed in the node table located to the left of the screen. The method get Table Data is used to retrieve all necessary node details from the database.

4) TopologyGui:

This module is the most complex as it handles all the components that are placed in the topology tab. There are three key methods used, make Nodes ,make Node Neighbors and paint Component. As there name suggests make Node creates the circles which represent nodes, make Node Neighbors create lines which represent links between a node and its neighbors and paint Component draws both of these sets of objects to the screen. Paint Component also draws, delete's, moves and resizes the selected areas (rectangles) to the screen based on users mouse clicks.

B. Database Modules

See class diagram in fig. 2.

1) Database Connection:

This module gives database access to other classes through its setup Connection method.

2) NodeTable/DataNodeTable/Neighbour Node Table:

Each of these modules provides access to their respective table in the database. Each of these classes contains a get TableById and getAllTable method which do as each name suggests. DataNodeTable however contains a method which gets a node's data when given the nodes name, called getNodeDataByName. This method contains multiple queries to achieve the required result.

C. Simulation Modules

See class diagram below in fig. 3.

Simulations's class diagram

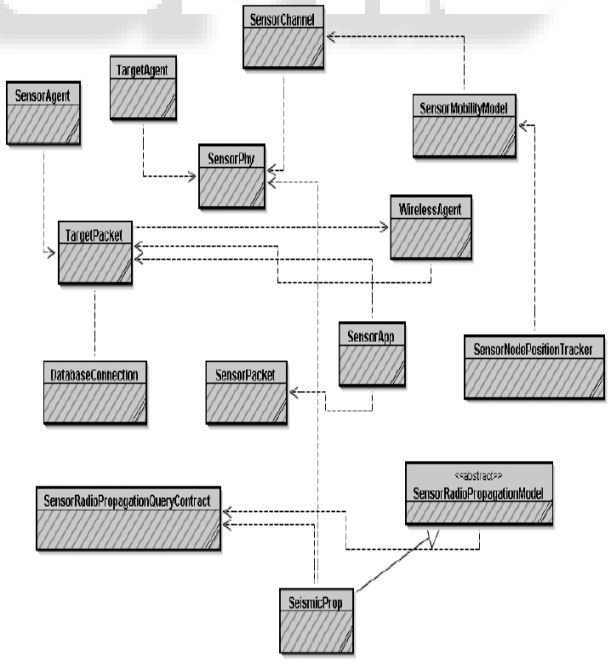


Fig. 3: Simulation's class diagram.

1) Sensor node:

This module represents a sensor node which is equipped with (1) a Sensor Protocol Stack which allows it to capture signals generated by target nodes through a sensor channel,

and (2) a Wireless Protocol Stack which enables it to send reports to the sink nodes through a wireless channel.

2) Target node:
Each target node will generate signals which are then sent to sensor nodes through the sensor channel. A target node includes (1) Target Agent Layer and (2) Sensor Physical Layer.

3) Sink node:
A sink node is built in a plug-and-play fashion using a Sensor Application Layer(SensorApp), an Interface Layer (Wireless Agent) with the Wireless Protocol stack: network layer (AODV protocol (drcl.inet.protocol.aodv.AODV)), MAC layer (MAC 802.11(drcl.inet.mac.Mac_802_11)), and physical layer (drcl.inet.mac.WirelessPhy)

4) Sensor channel:
The Sensor Channel receives a signal from a target node and sends a copy of that signal to all sensor nodes within the transmission radius of that target node.

5) Wireless channel:
Through the Wireless Channel, data from sensor is forwarded to the sink. In addition, the simulation also contains a Battery model, CPU model and Radio model.

D. Sensor Node Modules

See the class diagram below.

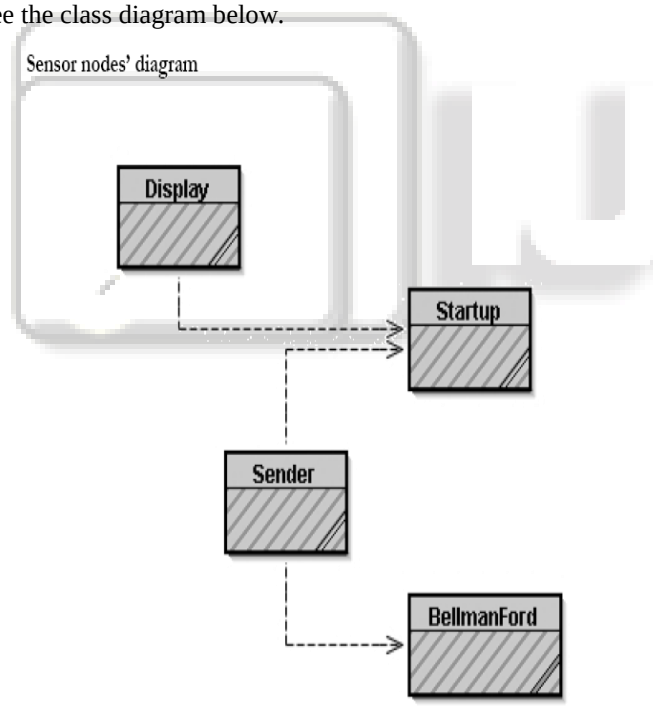


Fig. 4: Class diagram of Sensor nodes Module

1) Bellman-Ford:
The module contains the implementation of a multi-hop routing protocol, finding the shortest path using a distance-vector algorithm (Bellman-Ford)

2) Spot's code:
The code is deployed in each sensor node so that each node can communicate with each other as long as with the base station.

Sender: open connection to another node; read data from hardware and send that data through the opened connection.

Start-up: initialize and start the sender.

3) Base station's code:
The code is deployed in each sensor node so that each node can communicate with each other as long as with the base station.

Display: open connection to another node; read data through the opened connection and process that data.

Startup: initialize and start the sender.

VI. USER OPERATIONS

A. User Interaction with the System

The main way that users interact with the system is through the use of the visualiser, as it displays all the important data such as temperature and light retrieved from each node.

B. Viewing Statistical Data

The visualiser separates the main statistical data into page tabs, so it is easy for a user to view large amounts of data without being overwhelmed.

The visualiser tabs include:

1) Data:
Which shows the most current data (temp, light, location etc) retrieved from the database for each node? This is placed inside an adjustable table.

2) Topology:
This tab shows graphically the location of each node on a given map. Each node is represented by a red circle, with its id in the middle. A node's neighbors are represented by direct lines drawn from the node to each neighbor node. This tab is talked about further in the User interactions section, showing other ways a user can interact it.

3) Temperature Line graph:
This tab shows selected nodes in a line graph and how there temperature varies over time.

4) Light Line graph:
This tab shows selected nodes in a line graph and how there light varies over time. Further interactions with the line graph are talked about in the User interactions section.

VII. CONCLUSION

I feel that the Sun SPOT platform is ideally suited for the prototyping and developing of applications for wireless sensor networks.

First, there is enough memory, processing power, etc. on each SPOT that one can do interesting things locally; it is much easier to prototype when not constrained by tight resources.

Second, it is easy to extend the functionality of a SPOT by connecting sensors or actuators to the demo sensor board or by creating custom sensor boards.

Third, writing applications in the Java programming language is easier than using lower-level languages as C. Programming in the Java language also makes it possible to share code between applications on the device and on the desktop, and also makes it easier to integrate the wireless devices with the desktop.

Fourth, a program development environment that enables over-the-air deployment and debugging greatly simplifies the task of developing wireless network

applications. The Sun SPOT is a mid-level sensor device that enables exploratory programming of sensor applications. The software architecture is designed to be a framework for experimentation, encouraging the replacement and addition of new components, such as a mesh-radio stack.

REFERENCES

[1] Crossbow Technology, Inc., "Mote In-Network Programming User Reference" and "Mica2 Wireless Measurement System Datasheet," 2003.
[2] P. Levis et al., "Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks," Proc. 1st Symp. Networked sys. Design and
[3] Implementation, 2004, pp. 15–28.
[4] T. Stathopoulos, J. Heidemann, and D. Estrin, "A Remote Code Update Mechanism for Wireless Sensor Networks," Tech. rep. CENS-TR-30, UCLA, Center for Embedded Networked Computing, Nov. 2003.
[5] J. W. Hui and D. Culler. "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale," Proc. ACM SenSys 2004, pp. 81–94.
[6] S. S. Kulkarni, and L. Wang, "MNP: Multihop Network Reprogramming Service for Sensor Networks," Proc. IEEEICDCS 2005, pp. 7–16.
[7] P. Levis and D. Culler, "The Firecracker Protocol" Proc. 11th ACM SIGOPS Euro. Wksp., Leuven, Belgium, Sept. 2004.
[8] A SunSpot Simulator for Datagrams (bSPOTs), URL: http://www.cs.usyd.edu.au/~scholz/projects/Simulator/
[9] Bellman-Ford Algorithm, URL: http://en.wikipedia.org/wiki/Bellman-Ford
[10] J-Sim Home Page, URL:http://www.j-sim.org/
[11] MoteView Monitoring Software, URL: http://www.xbow.com/Products/productsdetails.aspx?sid=88
[12] Research – Research Areas – Sensor Nets / RFID - Intel®Mote, URL: http://www.intel.com/research/exploratory/motes.htm
[13] Sun Microsystems, URL: http://www.sun.com/
[14] Sun Microsystems Laboratories, URL: http://research.sun.com/
[15] SunSpotWorld – Home of Project Sun SPOT, URL: http://www.sunspotworld.com/
[16] Wikipedia. The Free Encyclopedia, Wireless Sensor Network, URL: http://en.wikipedia.org/wiki/Wireless_sensor_network.

