Computer Society International Design Competition 2006
Final Report

School:	Boston University
Country:	United States of America
Mentor:	Prof. Michael Ruane
	mfr@bu.edu

Team Members

Yaniv Ophir
yophir@bu.edu
Computer Systems Engineering

Vyas Venkataraman
vyas@bu.edu
Computer Systems Engineering

Joseph D'Errico
jderrico@bu.edu
Computer Systems Engineering

Andrew Hagedorn
achag@bu.edu
Computer Systems Engineering

1 Abstract

Although often neglected, urban ecosystems play a pivotal role in the health, economy, and quality of life of residents of urban areas. As one of the few areas in metropolitan settings where nature is protected and cultivated, parks are essential to preserve. Many city-dwellers value parks as a gateway to nature and a respite from their busy urban lives. In addition to the aesthetic appeal of parks, the grounds also serve as homes for many different species of plants and animals. However, human presence in parks has negative repercussions that can damage the ecosystem. One prominent visible sign of human impact is the presence of trash cans, often overflowing with litter. Furthermore, the trucks used to empty the trash cans are a major source of air pollution in parks. Park officials spend a great deal of time and money emptying trash cans. These expenses are not limited merely to the cost of equipment and personnel needed to empty the trash cans, but also to the effort expended to retrieve any litter that has fallen out of the cans. Since current trash can-emptying techniques entail static routes that assume every can is full, many unnecessary trips are made to unfilled trash cans, while trash cans that fill quicklyare not visited frequently enough.

The SmarTrash system enables quicker and more efficient emptying of trash cans to prevent the overflow of litter. Mesh-networked trash cans sense whether they are full by using two types of infrared (IR) sensors. Infrared light- emitting diodes (LEDs) and phototransistors examine a plane at the top of the trash can, and an IR distance sensor looks down into the trash can from the lid. Crossbow MICA2 motes are used to control the sensing hardware, provide wireless networking, and connect to an Ethernet Gateway board, which interfaces between the mesh network and a server. The server displays trash can status via a Google Maps-based web interface, making the status easy to identify while allowing for more efficient emptying of full trash cans. The server also stores the data in a database from where the data can be extracted and used to analyze long term trends by using various statistical analysis tools.

SmarTrash represents the first system enabling the centralized monitoring of trash cans. Utilizing data provided by SmarTrash, park officials can enhance efficiency by servicing only those trash cans known to be full. Moreover, SmarTrash will help parks nationwide save money on sanitation costs while simultaneously aiding in the maintenance of urban parks. Additionally, by harnessing renewable solar energy, SmarTrash avoids further exacerbating environmental problems.

Currently, the SmarTrash system exists as a fully functional prototype. Extensive tests of the system helped determine that acquisition and transmission of data occurred with an accuracy greater than 99%. The next step entails scaling up the system to deploy for long-term testing. Potential long-term future enhancements such as an automatic route finder, will provide park officials with additional tools to protect urban ecosystems.

2 System Overview

2.1 Problem

The inefficiencies in current trash can-emptying methods are harmful to the environment. Common garbage trucks are among the vehicles with the lowest fuel economy, averaging only 2.8 miles per gallon [1]. In parks and beaches where the main roads may be inconveniently distant, maintenance workers use vehicles such as Broyhill's Load-and-Pac. Although the Load-and-Pac is smaller than a conventional garbage truck, a discussion with Broyhill revealed that it burns about 3 gallons of diesel fuel per hour when loaded with trash and in heavy use. Since trash collection vehicles spend a significant amount of time idling at trash can sites, their exhaust tends to linger in the parks. Reductions in the time needed to empty trash cans as well as the number of trash cans to empty would certainly curtail the amount of pollutants emitted by these vehicles.

Trash collection is also an expensive and time-consuming process. In a SmarTrash survey sent to 50 municipal park authorities in the United States, 9 of 12 respondents conveyed interest in a system that would render trash can-related maintenance more efficient. Assessing the data from the survey, it was evident that large cities spend millions on trash collection; the City of New York Parks and Recreation, for example, estimates that they spend over US$8 million annually on trash collection in the Borough of Manhattan's parks alone. Even in a smaller city such as St. Paul, MN, US$5.33 is spent on each trip to a single trash can, regardless of whether it is emptied or not.

Furthermore, some trash cans require a higher frequency of emptying than others. Manually monitoring and emptying all the trash cans at the same frequency forces park officials to waste time in traversing large areas to determine the status of trash cans that may already be empty. Prior knowledge of full trash cans could reduce the time and effort expended in trash collection in all these examples.

2.2 Objectives

SmarTrash was created to meet the following five objectives. It should inform park services of the trash cans that need emptying without the need to visit each one. Data should be presented on a user-friendly GUI and stored in a format compatible with GIS software. The majority of communications should occur in a flexible manner over a wireless medium. SmarTrash should have minimal impact on its surrounding environment. Finally, it should be able to harness renewable solar energy to make it a self-sufficient system.

2.3 Performance Requirements

The following performance requirements were created to meet the objectives stated in Section 2.2.

Maximum Trash Can Spacing	100 m
Battery Lifetime	> 1 year
Data Acquisition & Transmission Accuracy	> 99 %
Operating Temperature Range	-10˚ C to 60˚ C
Data Acquisition Frequency	User configurable
GUI Portability	All contemporary desktop operating systems

Table 1. Project Specifications

2.4 Design Methodology

The design strategy for this project followed a six step methodology used by practicing engineers - research, define problem, plan solution, execute plan, verify implementation, and conclude [2]. In the research phase, the problems associated with litter in municipal parks were investigated and a survey was conducted. The data from the research phase was used in conjunction with standard problem definition tools, including a House of Quality matrix, Quad Chart, and a focus group. This clarified and finalized the definition of the problem in engineering terms. The solution to the problem had to be fully planned; this necessarily entailed the development of custom hardware and modeling flows of software. Execution of the plan involved selecting components, implementing circuits, and coding software modules. Major tasks were partitioned and assigned to different team members.

The team was organized as follows:

- Team Leader, Server Software, and PCB Layout – Yaniv Ophir
- Mesh Networking Software - Andrew Hagedorn
- Sensor Hardware and Software – Joseph D'Errico
- Power Supply Hardware and Power Management Software – Vyas Venkataraman

A timeline for completion of these modules was formulated using a Gantt chart. Three formal design reviews allowed for continual refinement of the solution. Portions of the verification stage were combined with the execution stage. Errors were minimized through the use of a modular build-and-test approach. Each component was first tested as a standalone unit. Finally, the components were integrated into a complete system and tested together.

2.5 Design Decisions

As the SmarTrash system was being created, several major design alternatives were considered before settling on the final design. An initial issue that surfaced was the choice of the networking algorithm. This is fully discussed in Section 3.6.6. Another major issue was the choice of sensor technology to detect trash. Strain gage based weight sensors were considered first. These had the disadvantage of requiring mounting on the base of the trash can, which would then require an electrical connection between the lid and the body of the trash can. As trash cans have to be opened regularly, this connection would be susceptible to wear and would render the system useless if broken. Ultrasonic sensors were considered next. This idea was eventually abandoned as the power consumption of ultrasonic sensors was prohibitively high. A system issue that greatly affected the final system design was the ability of the system to be able to wake up on a radio packet, as opposed to sleep achieved by powering down the radio transceiver completely. This allows the user to poll the system. This design decision traded off slightly longer battery lifetime in exchange for the ability for the user to be able to initiate a poll of the system.

2.6 Project Originality

The most common practice in park waste management is for park personnel to visit each trash can, determine if it is full, and then empty it if necessary. To the knowledge of the SmarTrash team, no park in the United States employs a centralized method of notifying staff when trash cans are full.

One technological update to conventional trash can services is the addition of self-compaction, as implemented in the Big Belly [3]. That system costs $4500 per can, almost twice

the price of conventional trash cans [4]. Moreover, the benefit of in-can compaction is diminished in an outdoor setting as many parks already use trucks with compaction in them to pick up trash.

The SmarTrash system demonstrates the use of unique technological achievements for a novel purpose. It is the first application of wireless mesh networked sensors to the field of trash collection. The system possesses redundant in-can sensors that can accurately relay the status of the trash can and are able to ensure that the system is not registering false data. The networking algorithm being used by SmarTrash is unique and was created to specifically balance the needs for accurate data transmission and low power consumption. The SmarTrash system also provides user interfaces in the form of a user friendly map and analysis ready data never before associated with monitoring trashcans. Since this application is unique, SmarTrash required the development of customized hardware and software, including an original sensing package. In keeping with the environmental aware design, the hardware designed for SmarTrash is RoHS compliant, i.e. is lead free.

3 Implementation

The system will be described by first describing the physical components, followed by the overall system architecture. Each subsystem in the architecture will then be elaborated.

Figure 1a. The MICA2 mote

Figure 1b. The MIB600 Ethernet Gateway

3.1 Hardware Components

3.1.1 MICA2 Motes

The MICA2 mote (Figure 1a) is a low-power embedded wireless device that runs on a 7MHz Atmel processor. It is powered by two 1.5V AA batteries and has various sleep modes to conserve battery power. Each mote has eight 10-bit analog-to-digital converter (ADCs) pins as well as eight general purpose input/output (GPIO) pins, both of which are accessible via a 51-pin connector. The wireless communication operates at 38.4kbps over a 433MHz channel. In an urban park setting, the measured maximum range of the MICA2 is 100 m.

3.1.2 MIB600 Ethernet Gateway

The MIB600 (Figure 1b) interfaces between the motes and the Internet. This board has a built-in DHCP client and a 10 Base-T connection, enabling it to acquire an IP address almost anywhere an Ethernet connection is present. A mote attached via the 51-pin connector provides the Ethernet Gateway with wireless communications capabilities.

3.1.3 Sensors

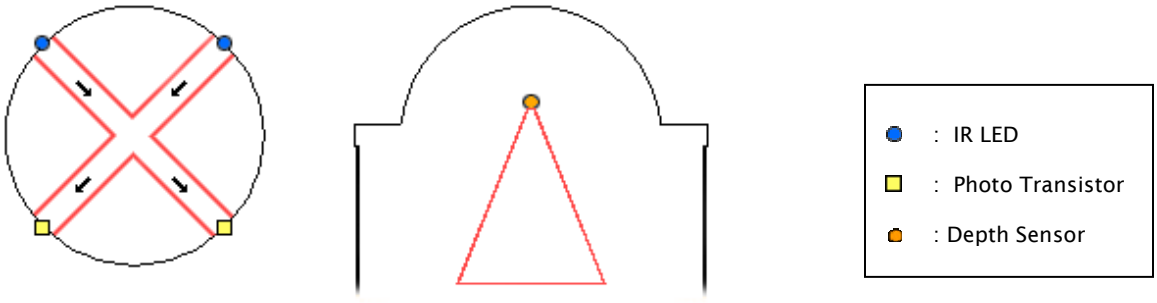

Figure 2. Left : The Planar Sensors in top cutout view of trash can.
Right: The Depth Sensor in front cutout view of trash can

SmarTrash uses two systems of IR sensors to detect the level of trash in a trash can (Figure 2). The Sharp GP2D12 Distance Measuring Sensor is mounted on the lid, facing down into the trash can. When activated, the GP2D12 uses IR reflection to determine the distance to the nearest object and outputs an analog voltage inversely proportional to this distance. This sensor can reliably measure objects from 80 cm to 10 cm . As this sensor is used to measure the depth of trash in the trash can, it is referred to as the ***Depth Sensor***. The second sensing system consists of two IR LEDs and phototransistor pairs, aligned across the rim of the trash can, forming a plane. Each phototransistor outputs an analog voltage proportional to the amount of IR radiation it detects. When the plane is obstructed, the phototransistor detects less IR radiation and will therefore detect a lower voltage. These ***Planar Sensors*** detect when trash has breached the trash can's rim.

3.1.4 Power Supply

While the MIB600 and the server require line power, the motes are powered by two 1.5V

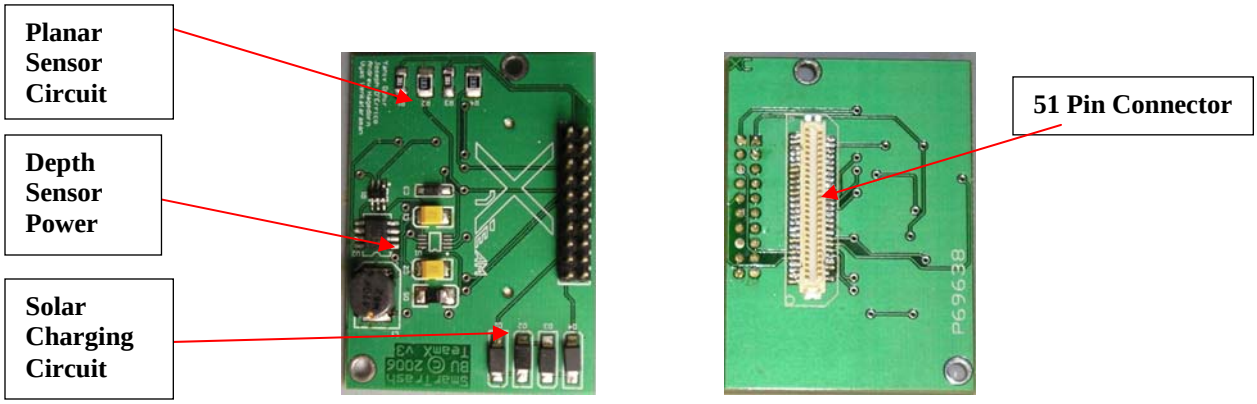

Figure 3. SmarTrash two sided PCB, version 2 RoHS compliant (Lead Free)

AA rechargeable alkaline batteries. To increase the operational lifetime of the system, each trash can unit is equipped with two 3V solar cells that trickle charge the batteries. The solar cells are in series with a set of diodes to regulate the charging current, allowing the solar cell trickle charger to provide a variable current in proportion to the voltage of the battery. When the battery is fully charged, the trickle charging circuit provides no current, ensuring that the batteries do not overcharge. As per design, the trickle charger is always active and will continue to charge the batteries until they reach their operating voltage, allowing SmarTrash to remain completely self-contained with no need for an external power source.

Power analyses were performed assuming the user polled the system two times per day, a number that is based on survey responses. The absolute minimum life of batteries in the motes is 5 months if no recharging occurs. A conservative estimate of the solar cells' benefits presumed that the trickle charging circuit provides an average of 2.5mA during the hypothetical 4 hours of usable sunlight each day. Under these conditions, SmarTrash has an operational life greater than two years.

3.2 Architecture Overview

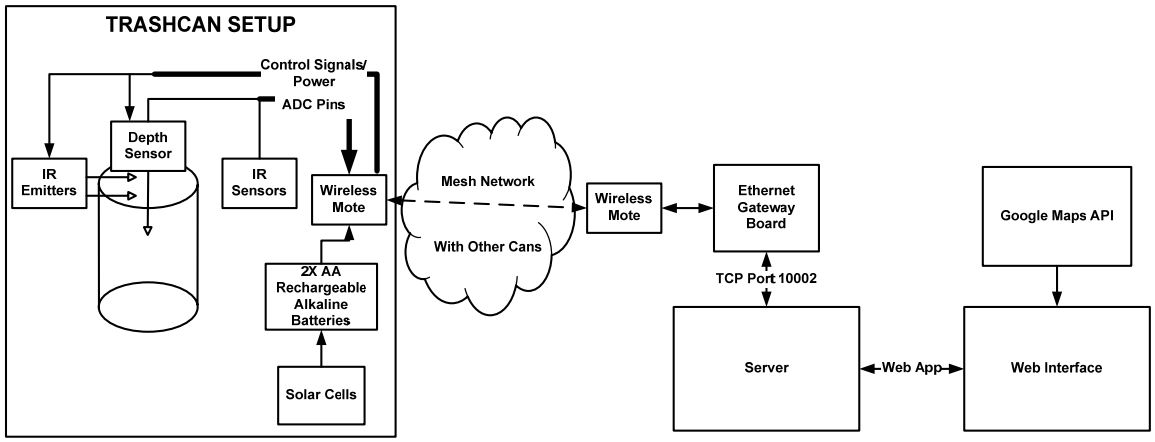

Figure 4. Architecture Overview

Each trash can is equipped with a two Planar Sensors, one Depth Sensor, a MICA2 mote, and solar panels for recharging the two AA batteries that power the mote. The mote communicates wirelessly with the Ethernet Gateway either directly or via a mesh network of motes in other trash cans. The Ethernet Gateway is accessed by a server that runs software to parse the messages received from the motes and display the motes' status on a map. Messages are passed in two directions through the network. Sensing data is sent from the motes through the mesh network to the Ethernet Gateway, which forwards the data to the server. Requests for the system to poll its status are sent from the server through the Ethernet Gateway and propagate through the mesh network. These signals can either originate as part of a scheduled system update or be initiated by the user via the Web interface.

3.3 Software Languages and Environments

The mote software runs on the TinyOS operating system, which was developed for use in limited resource network systems. It is released under the Intel Open Source License. All mote software, including TinyOS, is written in the nesC language.

The operating system of the server is Ubuntu, a distribution of GNU/Linux. The server runs Apache2 to handle web page requests and a mySQL database to store static can information. Software running on the server is written in Perl, while queries to the database are written in SQL. The web GUI uses the Google Maps API [6] and AJAX. Various tools created to facilitate and augment intermediate testing were written in rapid application development environments such as C#.NET and Java.

3.4 Power Management

In addition to the solar powered trickle charging hardware, the mote has power regulation software that allows it to enter low power sleep. The low power sleep mode on the mote is enabled when the mote's task queue is empty and all high speed interrupts are disabled. In this mode, the mote's radio is set to function at a 1% duty cycle, which allows it to drastically reduce its current consumption. The motes remain capable of receiving packets sent to them, and when a wake-up packet is sent, the mote sets the antenna to operate at a 100% duty cycle and begins the sensing cycle. While it is possible to completely turn off the radio to achieve even lower power consumption, this approach was not incorporated as guaranteed wake-up synchronization would negate the calculated 4% increase in battery life.

3.5 Sensing

3.5.1 Sensing Algorithm

When the mote is prepared to sense, it provides power to the sensors. For each sense cycle (Figure 5), the mote gets two readings, 110 s apart, from each sensor. These redundant readings allow the sensing decision algorithm to compensate for potential malfunctions in hardware. Between each reading, power to the sensors is turned off to minimize the current draw on the system. For each reading, the mote samples the IR phototransistors with the IR LEDs both on and off. Ambient infrared level is compensated by computing the difference between successive samples. The differences are averaged to give the final value for that reading. The Depth Sensor is unaffected by ambient infrared, so no special consideration is needed for this sensor. Only one sample is taken per reading of the Depth Sensor. The six readings are not interpreted on the

motes themselves; they are sent over the mesh network to the server, where they are used to make a decision about the state of the trash can.

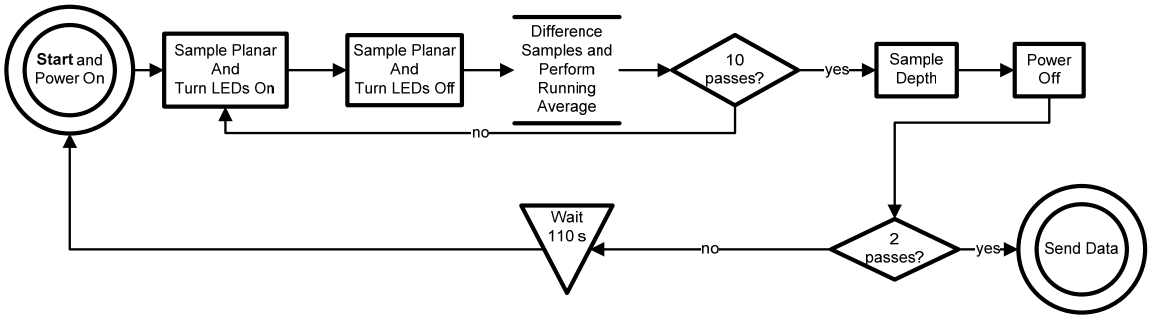

Figure 5. Flowchart of sensing cycle

3.5.2 Decision Algorithm

The intended object of detection is the level of trash. The Depth Sensor threshold (DEPTH_THRESHOLD) is calibrated such that Depth Sensor triggers if there is an object less than 50cm from the rim of the trash can. The Planar Sensor threshold (PLANAR_THRESHOLD) is calibrated such that the Planar Sensors trigger if an object blocks more than 75% of the beam. A transient could cause a false trigger; a person throwing trash away at the exact moment a sensor is sampled could be detected. Therefore, two readings spaced 110 s apart must both indicate the presence of trash. By default, the sensor cycles are repeated every 12 hours, though the interval can be changed. Also, sensor cycles can be manually initiated via the web interface.

```
IF (Planar1_Reading1 < PLANAR_THRESHOLD) AND
       (Planar1_Reading2 < PLANAR_THRESHOLD)
THEN Planar1 sees Trash

IF (Planar2_Reading1 < PLANAR_THRESHOLD) AND
       (Planar2_Reading2 < PLANAR_THRESHOLD)
THEN Planar2 sees Trash

IF (Depth_Reading1 > DEPTH_THRESHOLD) AND
       (Depth_Reading2 > DEPTH_THRESHOLD)
THEN Depth sees Trash

IF Planar1 sees Trash AND Planar2 sees Trash
THEN
{
       IF Depth sees Trash
       THEN Trash can is "Full"
       ELSE Trash can "Needs Maintenance"
}
ELSE
{
       IF Depth sees Trash
       THEN Trash can is "Almost Full"
       ELSE Trash can is "Empty"
}
```

Figure 6. Pseudo code for the decision algorithm

Both Planar Sensors must detect trash or the algorithm does not determine that the trash can is full. This is to compensate for any failure that may occur in a Planar Sensor, which would give a low voltage reading and result in a false positive. By requiring both Planar Sensors to detect trash, the algorithm reduces false positives.

If all sensors detect trash, the trash can's status is set to "Full." Since the Depth Sensor detects trash 50 cm from the rim and would presumably detect trash before the Planar Sensors, a Depth Sensor reading above DEPTH_THRESHOLD sets the trash can's status to "Almost Full." The trash can's status is set to "Empty" if no sensors detect trash. If the Depth Sensor fails, it gives a low voltage, or "Empty,", reading regardless of any trash that may exist. If both of the Planar Sensors detect trash, trash must exist within the range of the Depth Sensor. Therefore, if the Planar Sensors both detect trash, and the Depth Sensor does not detect trash, then the Depth Sensor has failed. In this situation, the trash can is given a "Needs Maintenance" status.

3.6 Mesh Networking

3.6.1 Introduction to Mesh Network Algorithm

The mesh networking algorithm was specifically developed for SmarTrash. This application required that the algorithm have several properties. Data must be guaranteed to reach its destination, either directly or by taking multiple hops over several motes. The network must be able to heal; in the event that a mote fails or cannot communicate with its neighbors, the network must be able to find an alternate route if one exists. All networking must be implemented with minimal memory use as the wireless motes are a restricted memory environment.

Conceptually, the mesh network is a tree where all data flows to the root. In this tree, routing is accomplished by each mote knowing the next mote in the tree that is closer to the root, its parent. In this implementation, each mote needs only to store its parent's 16-bit address instead of a full routing table which contains every mote in the network. Not only does this minimize the memory overhead of the network, but it also prevents each mote from having to update its routing table every time the network heals. When a mote wants to communicate, it attempts to send a data packet to its parent and then listens for its parent to forward the packet. Since all messages are broadcast using the mote's omnidirectional antenna, the forwarded packet can be heard by the original sender mote and acts as an implicit acknowledgement. If the mote does not hear this acknowledgement, then the mote resends its packet, allowing any nearby mote to forward it through the network. This guarantees that if there is a path to the root, the packet will be forwarded there; this process also heals the network to compensate for the loss of a mote. To ensure that the healing is permanent, the mote that has lost its parent will update its parent address with the address of the mote that receives its data packet.

3.6.2 Network Components

The network is comprised of motes that have been programmed to be one of three types. ***Data Motes***, which are present in the trash cans and collect data from their sensors, are the most common type of mote in the network. After data collection, Data Motes transmit their data and forward any data that is sent to them. Similarly, the ***Extender Motes*** also forward any data that they receive. Extender Motes have the same power hardware and store the same networking information as Data Motes, but are neither located in trash cans nor attached to sensors. Extender Motes can optionally be powered by external sources. If two Data Motes are placed farther apart than their range allows, an Extender Mote will bridge the gap. From the viewpoint of the networking protocol, Extender Motes are a subset of Data Motes and are treated identically. A single ***Base Mote*** is the third kind of mote in the network. The Base Mote is positioned at the root of the network and forwards all data from the Data Motes to the server.

The Base Mote also relays control signals from the server to the network. To accomplish this, the Base Mote is connected to an Ethernet Gateway, which allows the server to establish a connection through which the Base Mote can transfer its data.

3.6.3 Data Packets

Data Field	Length	Meaning	Values
Type	8	Data or Wake up Packet Type	0x21 or 0x2C
Source	16	Address of sending mote	0x0000-0xFFFF
Destination	16	Address of destination mote.	0x0000-0xFFFF
Hop Count	16	Hop count of the sending mote	0x0000-0xFFFF
Data Frame 1	8	Data head delimiter	0x42 (ASCII B)
Data Frame 2	8	Data head delimiter	0x45 (ASCII E)
Data Frame 3	8	Data head delimiter	0x47 (ASCII G)
ID	16	Mote of data's origin.	0x0000-0xFFFF
Voltage 1	16	First reading of Planar Sensor 1	0x0000-0x03FF
Voltage 2	16	Second reading of Planar Sensor 1	0x0000-0x03FF
Voltage 3	16	First reading of Planar Sensor 2	0x0000-0x03FF
Voltage 4	16	Second reading of Planar Sensor 2	0x0000-0x03FF
Voltage 5	16	First reading of Depth Sensor	0x0000-0x03FF
Voltage 6	16	Second reading of Depth Sensor	0x0000-0x03FF
Data Frame 4	8	Data tail delimiter	0x45 (ASCII E)
Data Frame 5	8	Data tail delimiter	0x4E (ASCII N)
Data Frame 6	8	Data tail delimiter	0x44 (ASCII D)

Table 2: Data Payload Contents

The TinyOS operating system limits the size of the packet data payload to 29 bytes. All network packets have the same format (Table 2). All packets types require the first four fields for route control and forwarding. A value of 0xFFFF in the Destination field will broadcast the packet to all motes. The data frames that follow are delimiters for the parsing software on the server. It allows the software to clearly find the ID and voltages.

3.6.4 Collision Avoidance

Each mote makes multiple attempts to send each data packet, stopping if acknowledged before an upper limit is reached. To arbitrate through a collision, each mote implements a unique delay timer. The delay is calculated using a linear function of the mote's unique local address. Any communication from its parent, an indication that the parent is functional, forces the mote to increase its delay time. This prevents excessive traffic on the network while its parent is busy.

3.6.5 Detailed Algorithm

All motes hold three pieces of networking information: their unique local address, their hop count from the root, and the address of their parent. When the system is reset, the routing information present on each mote is shown in Table 3 below:

Type of Mote	Local Address	Parent Address	Hop Count
Data Mote	Unique	0xFFFF (infinity)	0xFFFF (infinity)
Extender Mote	Unique	0xFFFF (infinity)	0xFFFF (infinity)
Base Mote	0x0000	NA	0x0000

Table 3: Network Initialization

Initially, the base mote broadcasts a wake-up packet. Sleeping motes receive this packet, causing them to exit low power sleep mode. The mote updates its parent to the source of the packet, sets its hop count, increments the hop count of the packet, and then forwards the packet. Motes that are awake drop this packet to prevent the network from infinitely forwarding wake-up packets. Thus, as the wake-up packets propagate through the network, they set up the initial network tree.

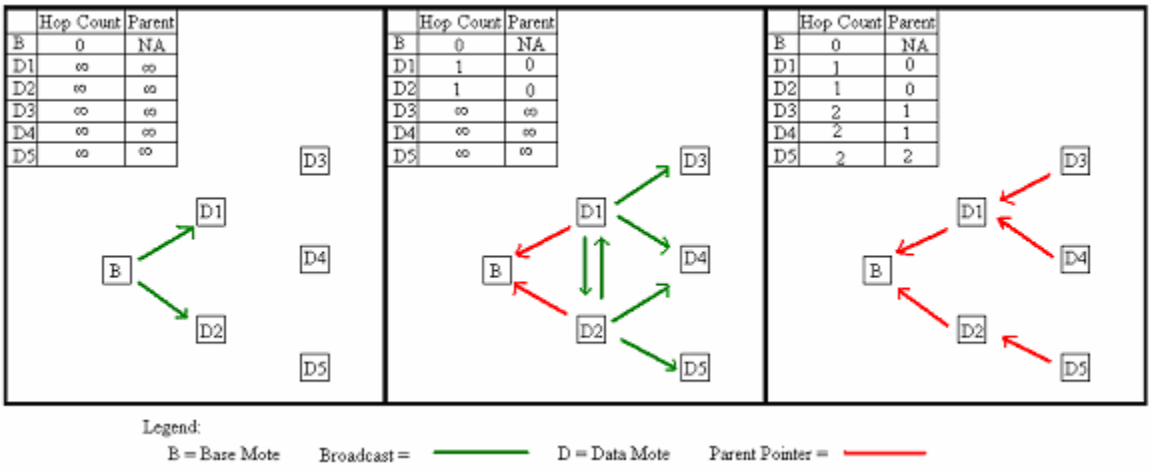

Figure 7 : Network Initialization

Figure 7 shows the initial setup algorithm. The Base Mote, B, broadcasts a packet so that all motes within range will receive the packet and update their routing information. In the figure, Data Motes D1 and D2 are within range to receive the packet, which they then broadcast to all motes within their range. Since D1 and D2 are already awake, they will disregard the packets that they receive from each other. In the next tier of the tree, Data Mote D3 receives a packet from D1 and updates its data; D5 does the same with its packet from D2. D4 receives a packet from both D1 and D2, but since there is a delay before sending based on local address, it will receive the packet from D1 first. D4 then updates its information atomically to prevent race conditions, and when it receives the packet from D2, it is dropped because it does not have a better hop count. This leads to a routing tree as seen in the third panel of Figure 7, which illustrates the optimal route for each mote.

While meshing, there are three basic operations that a Data or Extender Mote can perform: sending the data of the mote, forwarding the data of another mote, and waiting for the mesh cycle to time out. When sensing is complete, each Data Mote attempts to send its data packet to its parent by following the resend algorithm (Figure 8). After each attempted send, the mote listens to the medium for all network activity and waits for its parent to forward its packet. The action of the parent forwarding is an implicit acknowledgement of the data packet. If the Data Mote does not hear its parent forward the packet in a set amount of time, then the mote attempts to resend the packet to its parent. It repeats this process eight times with an increasing timeout between data resends. After this point, the Data Mote attempts to heal the network and find a new parent. This is accomplished by resetting its hop count to logical infinity (0xFFFF) and its parent to the broadcast address. It then resends to the broadcast address so that any mote within range will receive the packet. Though this creates a temporarily sub-optimal routing path, it will ensure that the data is able to get to the Base Mote if a path exists. The sub-optimal path is only

temporary because if the mote ever receives a packet with a better hop count, it will change its parent address and hop count. This can also lead to redundant copies of packets, but this issue is handled on the server.

```
Resend( )
  IF ( Resend Flag == TRUE)
  THEN
  {
        IF ( Resend Counter > 8) AND
                ( Resend Counter < 16 )
        THEN
        {
                Parent = Broad Cast Address
                Hop Count = Infinity
                Call Send( Data )
                Call Resend( )
        }
        ELSE IF ( Resend Counter < 8 )
        THEN
        {
                Call Send( Data )
                Call Resend( )
        }
        ELSE
        {
                Resend Flag = FALSE;
                Call Time Out( )
        }
```

Figure 8: Pseudo-code for Resend algorithm

Once a Data Mote's own data packet has been implicitly acknowledged by another mote, the mote enters a time out phase in which it waits for network traffic to stop before re-entering the low power sleep mode. When the mote receives a packet that is addressed to it or to the broadcast address, the mote exits the time out phase and attempts to forward the packet. The forwarding process follows the same resending algorithm (Figure 8) that the mote uses for its own data. When the forwarded packet is acknowledged, the mote re-enters the time out phase. This ensures that the mote is awake for all network traffic and allows all data to propagate through the network.

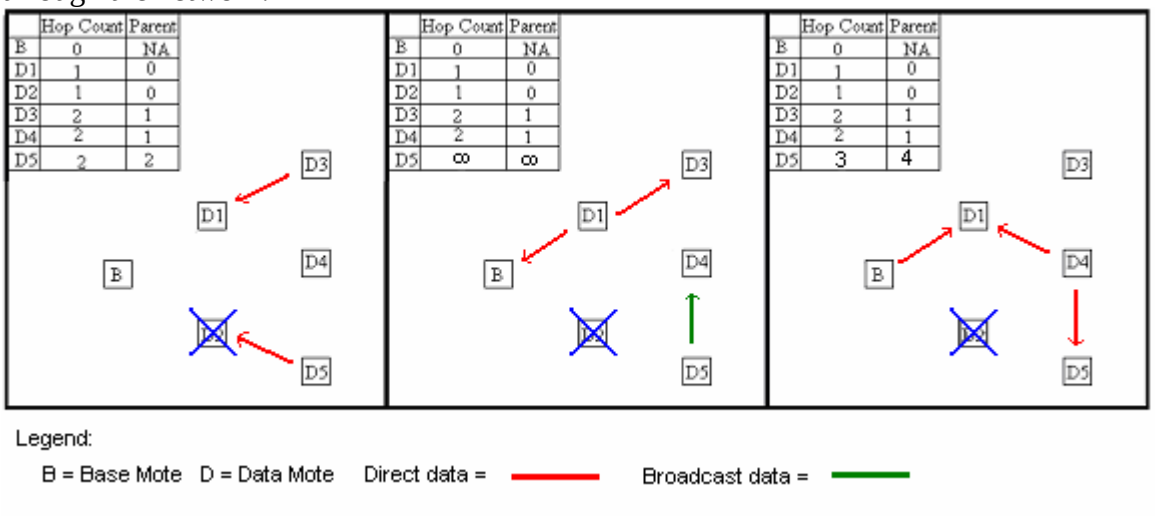

Figure 9 : Mesh Network Data Propagation and Healing

In Figure 9, the mote D2 is absent from the network, D3 and D5 are attempting to send their own data, and the routing table is set up similarly to Figure 8. When D3 and D5 send data packets to their parent, D1 receives the data packet; however, since D2 is out of the network, it

does not receive the packet from D5. In the second panel, D1 forwards the packet to the base station and D3 receives implicit acknowledgement when it hears its data being forwarded. D5 will eventually attempt to heal the network by broadcasting its data packet. D4 will receive the packet and forward it to its parent, D1. As with D3, D5 will receive the acknowledgement when it hears its packet broadcast to D1. In the third panel, the activity at D1 is variable. Though the base has forwarded its data to the server, it rebroadcasts the packet it received as an acknowledgement. D1 will either receive both the packet from the base mote and the packet from D4, or it will receive only one of them. In either case, D1 or D4 will not receive the implicit acknowledgement and resend to its parent, allowing the data to propagate to the base station.

3.6.6 Networking vs. Battery Life Trade Offs

The need for the mesh network to transmit all of its data is paramount. As discussed in Section 3.4 the motes spend a majority of their time in a low power sleep mode, which allows them to maximize battery lifetime. However, in the event that a mote can find no route to transmit its data, it must conserve battery power while waiting for some future time when a route may exist. In order to balance these two needs, the networking algorithm makes the following trade off. As long as the mote receives new data to forward and can effectively forward this data, the mote will stay awake, favoring accuracy of data transmission. When there is no more data to forward, or when it is unable to find another mote to forward the data to, the mote saves battery power by going into low power sleep mode after a set amount of time.

3.7 Server Applications

3.7.1 Server Initialization

Sending the position of each mote in each transmission is wasteful as this data is static. As each trash can has a unique local address, the mySQL database on the server will be able to correlate the ID of the mote to its geographical location. This database is created during

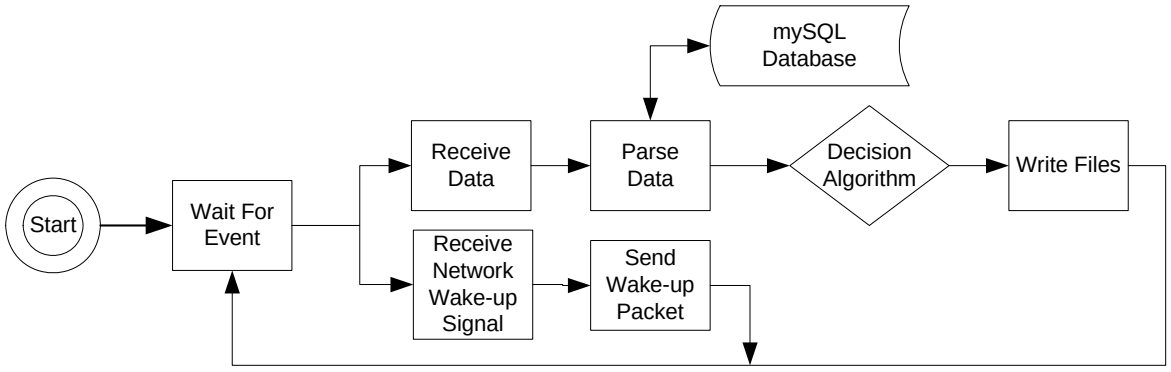

Figure 10 : Flowchart for Server Software

installation of the system by using a GPS receiver or geocoding software to determine the trash can's coordinates and then making an entry for that mote identifier in the mySQL database. This database also stores a log of the status of the system over time.

3.7.2 Server Software

The server software, SmarTrash Transmitting And Receiving Daemon (STARd), handles communication with the mesh network and stores incoming data to the hard disk. It was written in Perl to capitalize on the language's specialized parsing functions. STARd opens a TCP connection to the Ethernet Gateway and, once connected, waits for one of two events to occur. One event, a network wake-up signal, can either be user-initiated through a web interface, or can be set to automatically occur at a specific time. When this signal is received, STARd sends a packet to the Ethernet Gateway that wakes the motes and initiates the mote sense cycle. Additionally, STARd generates a unique run ID and writes it into the mySQL database along with the time of the signal. The other event occurs when data is received from the Ethernet Gateway. In this scenario, STARd extracts the 16-bit voltage readings and the mote ID, discarding the excess networking overhead.

STARd uses the mote ID to query the mySQL database to retrieve a trash can's latitude and longitude. The voltage readings and respective geographic locations are placed in a hash table keyed by the mote ID (Figure 11). Once the hash table has been filled, each trash can's status is determined by the decision algorithm (Section 3.5.2) and saved in the hash table. As a large number of packets can arrive in a short amount of time, the packet data is buffered by adding it into a hash table. STARd waits until no packets have been received for 300 seconds to write the contents of the hash table to hard disk, preventing excessive writes.

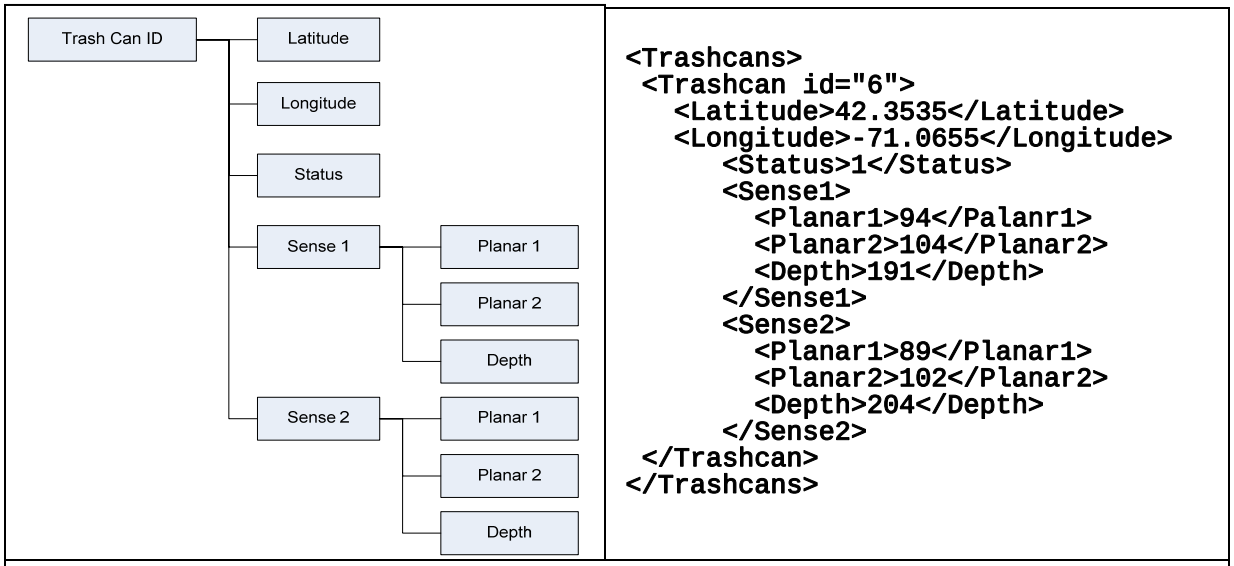

Figure 11 : Top Left: Trash can data structure; Top Right: SmarTrash XML; Bottom: Google Maps compatible XML

Trash can status information is stored in two XML files. The first includes all of the data; this file is used for logging, debugging, and as a data transfer module to other programs, such as ESRI ArcGIS mapping software. The second follows guidelines specified by Google Maps. This file stores only the locations and status of the trash cans.

Additionally, the status of each trash can after a run is stored in the mySQL database for data logging purposes. The database has two tables, one that has the run ID and the time the sense cycle was initiated, while the other contains a list of the trash can IDs, trash can statuses, and the run IDs. The second table contains trash can status from each run, so all the previous history of the system is preserved. The server software also has a tool that allows the user to output the values from the table into a tab delimited text file that can easily be imported into a spreadsheet program for further analysis. At the end of every sense cycle, STARd marks its data as old data, and when on the next sense cycle, new data is received from a trash can, that trash can's data is marked new. If data is not received for one trash can on a given run, then its last known status is displayed along with a small red cross on its icon to indicate that no data was received from that trash can on this sensing cycle.

3.8 Web Interface

The user interacts with SmarTrash via a web browser. The GUI is built using HTML, AJAX, and Version 2 of the Google Maps API. When the user first opens the web site, a map is created using the Google Maps API. An AJAX script then loads and parses trash can information from an XML file, and updates the status of each trash can on the map (Figure 12). The web interface also contains a 128-bit SSL secured area where, after password-based authentication, the user can initiate a poll of the trash cans' status.

A web browser interface was chosen over a standalone program to ensure portability and user-familiarity. Moreover, many potential users have had exposure to online maps. The Google Maps API makes it easy for users to manipulate the map, move around the field, and zoom in and out in familiar ways.

Figure 12 : Screen shot of Web interface

Icon	Trash can Status
	Empty
	Almost full
	Full
	Needs maintenance
	Empty, old data
	Almost full, old data
	Full, old data
	Needs Maintenance, old data

Table 4 : Map Icon Legend

3.9 Practical Considerations

3.9.1 Economic Viability

A preliminary cost analysis suggests that SmarTrash is economically viable. To demonstrate this, the SmarTrash system can be compared to the cost of current trash collection systems in parks. A typical, full-size (55 US gallon) outdoor trash can costs between US$400 [6] and US$2500 [6]. SmarTrash infrastructure expenses, which include the web server and Ethernet Gateway, can be as low as US$325 (Table 5b). Subsequently, the additional hardware per trash can cost approximately US$200 (Table 5a). Further, the US$200 in hardware does not factor in economies of scale, in which case the price of mote hardware could drop by a factor of 10 [7].

Description	Cost in US$
Custom PCB	33.00
Batteries (2x AA)	3.60
MICA2 mote	125.00
GP2D12 Infrared Depth Sensor	11.50
Solar Cell (2x 3V solar cells)	24.50
Small components	15.14
Total	**212.74**

Description	Cost in US$
MIB 600 Ethernet Gateway Board	300.00
Server Running Linux	25.00
Total	**325.00**

Table 5a (Left) : Hardware cost per trash can
Table 5b (Top) : Infrastructure hardware cost

Studies of the system's economic feasibility also showed that the system had a payback period of 28 months. The calculations used very conservative estimates and assumed a cost of US$200 per trash can and an attrition rate of 5% of total cans per year. The study was for a 100 can deployment, in a usage scenario where 5% of total monthly trips were saved and each trip cost US$5.33 per can, regardless of whether it was emptied or not. It is projected that when the costs per can fall to US$100, with all other parameters constant, the system will have paid for itself in less than 14 months. The figure for the attrition rate provides correction for vandalism and loss of trash cans.

3.9.2 Scalability

The system has been tested using four motes. However, based on TinyOS limitations, the system is capable of supporting up to 65,000 motes per Ethernet Gateway.

3.9.3 Physical Construction

The SmarTrash system is designed for seamless integration into a customized injection-molded plastic lid that fits onto a 55-gallon steel trash can. The lid features weather-proof housing for all electronics, including channels for wiring, while internally mounting the antenna protects it from vandalism. With the exception of the batteries, which are accessible via a locked compartment, all internal components are sealed within the housing to prevent theft. The hardened solar cells are mounted externally and appropriately positioned to maximize sun exposure. Although the plastic lid and low mass of electronics provide some protection against impact damage that is expected from service operations of emptying cans, or minor vandalism, the electronics are attached using rubber mountings which further reduce shock. As a final precautionary measure, the lid is secured to the trash can to prevent its accidental or malicious removal. As is evinced by the CAD schematic of the proposed lid (Fig. 13), the Planar Sensors

are housed in the grey ring. The Depth Sensor is mounted in such a way that it looks through the hole in the green housing. The sensors are integrated into their housings and require little maintenance, as the entire assembly can be easily replaced. The blue compartment contains the mote, antenna, PCB and the battery pack. The entire lid is created out of injection- molded plastic and thus the antenna can be housed internally, preventing vandalism while retaining the ability to transmit at close to its full range. The various colored housings snap into place, allowing for ease of assembly while ensuring that the electronics are well protected.

The prototype system has not implemented the expensive injection molded plastic lid, but has used a standard steel lid. All testing was with the steel lid and an external quarter wavelength antenna. Performance with the plastic lid should be as good as or better than the prototype.

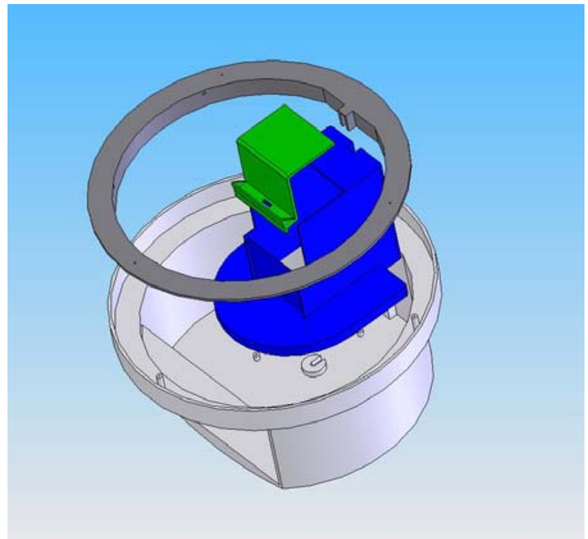

Figure 13 : Breakaway view of Plastic Lid

4 System Verification

4.1 Tools

Two tools were developed to aid in the design, debugging, and verification of the main design effort.

4.1.1 Mote Interpreter

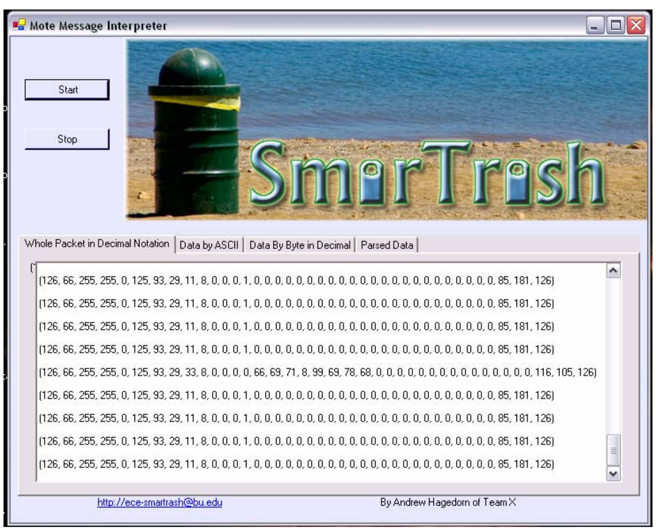

Figure 14 : Screen shot of mote interpreter software

A mote message packet interpreter was developed using C# .NET Framework to aid in the debugging of the network. The program connects with the Ethernet Gateway, parses packets into several formats, and outputs the data to both the screen and a file. This allowed the programmer to see the packets that were being sent between the Base Mote and the Data Motes in the network.

4.1.2 Packet Injector

The packet injector was developed to test the sleep and wake-up functions of the mote. The tool, based on a TinyOS program, is written in Java and enables the user to inject a packet into the mote running on the Ethernet Gateway. This packet is then broadcast to all the nodes, signaling that the mote should wake-up and begin its sensing cycle.

4.2 Incremental Testing

The Depth and Planar sensors were tested extensively before being integrated with the mote software. First, a mock construction of the trash can rim was created to test and revise the sensing methods. When the results demonstrated that the system could effectively discern the presence of an object, software was written to control the sensors. This was also tested separately from the entire SmarTrash system, using the MICA2's LEDs to indicate the sensor status.

The sleep software was tested by measuring the current consumption of the mote hardware and by verifying its ability to interpret the wake-up and sleep packets correctly.

The mesh network was tested both as a standalone system and as a component integrated with the sensing and sleep software. The verification process encompassed three separate tests: single-hop data transmission, multi-hop data transmission, and high traffic testing. Single-hop data transmission tests were conducted with a single mote communicating with the Base Mote, while multi-hop data transmission tests were conducted with a set of 4 motes. Communications were verified with a mote two and three hops away from the Base Mote. To simulate high traffic, the motes were placed within close proximity (1 m) of the base station so that the probability of collisions was increased. In all cases, the percentage of data packets received by the Base Mote was 100%.

Combined testing of the networking, sensing, and sleep software focused less on the capabilities of each of the individual components and, instead attempted to demonstrate that the three SmarTrash mote software pieces functioned as a unit. The initial round of tests addressed single-hop transmission of sensor data to the Base Mote. The goal of this round was to observe the voltages transmitted to the Base Mote when the sensors were blocked in different configurations. Since the desired range of voltages were known a priori from previous testing of the sensors, the concordant values received by the Base Mote were found to be correct.

The web interface has been extensively tested with all leading browsers and is compatible with Firefox, Internet Explorer, Netscape, Opera, Camino, and KHTML/Safari.

4.3 Full System Testing

The SmarTrash system was also fully tested with all components integrated. Results were determined by observing the map interface. The system contains two major variables: – the network and the sensors. The large number of possible distributions of trash within a trash can creates a large number of circumstances to test. Sample trash used to test the accuracy of the system was a heterogeneous mix of paper and plastic objects. The system was tested with the following four levels of trash, which encompass essentially all working trash can statuses:

- 10 cm from the bottom of the can
- 50 cm from the rim of the can
- 30 cm from the rim of the can
- 1 cm over the rim of the can

There are four possible configurations of the mesh network using the four motes available to the SmarTrash team. The following network configurations, using four motes, were tested for each trash level:

- [(Base)],[(1 Hop)],[(1 Hop)],[(1 Hop)]
- [(Base)],[(1 Hop)],[(1 Hop)],[(2 Hops)]
- [(Base)],[(1 Hop)],[(2 Hops)],[(2 Hops)]
- [(Base)],[(1 Hop)],[(2 Hops)],[(3 Hops)]

One mote acted as a Base Mote, one mote was attached to the sensors, and the remaining two motes sent an arbitrary valid voltage values. The mote attached to the sensors was in every possible position in each permutation.

From these extensive tests, it was determined that the system was accurate in 99.3% of the total results. "Accurate" means here that each mote determined correctly the trash can's empty/full/disabled status and successfully transmitted this data to the Base Mote. The 99.3% represented 1 failures out of 140 trials.

5 Summary

5.1 Conclusion

The SmarTrash system exists as a fully functional prototype. The system consists of a custom-designed circuit board (Figure 4) connected to sensors and MICA2 motes integrated into a 55-gallon outdoor trash can. The trash can lid currently in use is a steel hemispherical lid, although future models will employ a custom-built plastic lid. The antenna and solar cells are mounted externally on the lid, while all other electronics are housed internally in a ruggedized plastic box and attached with shock-absorbing rubber grommets and washers to the lid. The system is able to detect the status of a trash can and transmit the sensor readings over single or multiple hops to the base station, where the server software can extract the data and update a Google Maps-based GUI to reflect the new information. The server software also produces the data in formats that can be converted for use in ESRI ArcGIS. Additionally, the web interface allows users to manually initiate a poll of system status. The motes enter low power sleep mode and the solar cells recharge the mote batteries, permitting self-contained operation. This complete system has been shown to successfully meet design specifications and functional requirements. SmarTrash is able to store data from each run for a long term allowing the user to perform statistical analyses to discern long term trends in the trash cans' statuses, that could lead to more efficient positioning of trash cans.

Additional analyses of the economic viability and scalability show that SmarTrash lends itself to large-scale deployment in urban parks. A system like SmarTrash will simultaneously pay for itself and protect the environment. Using conservative estimates and data collected from the SmarTrash survey, a payback period of two years is estimated. Most importantly, SmarTrash will enable park officials to efficiently schedule trash pickup, reducing emissions from garbage trucks and allowing more time for other environmental projects, while still effectively preventing the overflow of trash cans.

5.2 Future Work

The next stage of work involves the finalization of designs for and construction of the custom plastic lid. Schematics for this lid have been created in CAD software to allow for mass-production. Once the plastic lid is fabricated, long-term outdoor testing of the SmarTrash system can commence. When these tests are successful, more lids can be manufactured to allow for deployment on a larger scale.

The server-side processing of information opens the door for the server to be able to provide additional services to the user, such as approximate route finding. Approximate route finding will add an overlay on the map, showing a reasonably good path a worker could employ to empty the trash cans. An existing best-case algorithm with reasonable computation overhead would be used.

6 References

[1] B. Siuru, “New Study Makes Strong Case for Natural Gas Garbage Trucks (Alternate Fuels)”, 2003 Nov 1, http://www.highbeam.com/library/docfree.asp?DOCID=1G1:111463795&ctrlInfo=Round19%3AMode19b%3ADocG%3AResult&ao=

[2] J. Stadtmiller, Project Management and Design, 2nd ed. Prentice Hall, 2003.

[3] A. Christopher, “BigBelly Has Appetite for Trash”, 2005 Mar 20, www.wired.com/news/planet/0,2782,66993,00.html

[4] CSN Supply, 2005 Oct, http://www.csnsupply.com/,

[5] Google Maps API, 2006 Apr, http://www.google.com/apis/maps/

[6] Trashcancentral.com, 2005 Oct, http://www.kitchensource.com/trash/

[7] A. Ricadela, “Sensors Everywhere”, 2005 Jan 24, http://www.informationweek.com/shared/printableArticleSrc.jhtml?articleID=57702816