Wireless Social Networking

Lee N Dobryden
Wireless Social Networking

An Honors Thesis (HONRS 499)

By

Lee N Dobryden

Thesis Advisor

Dr. Paul V. Gestwicki

Ball State University

Muncie, Indiana

July 2008
Abstract

The world of social networking has become a huge phenomenon. Extending these social networking sites past the extent of the Internet to mobile ad hoc networks has not gained such popularity. The possibilities of making social networking with electronic devices with a local ad hoc network could be quite amazing. The purpose of this research is to look into the possibilities of what a social network using mobile networking might look like.

To find out what the possibilities are I have put together a prototype. Using the SunSPOT device built by Sun Microsystems, I have created a program that exchanges information over an ad hoc network. This program uses the SPOT device to store, transmit, and receive information from other SPOT devices when within range. A user can upload information onto their SPOT device and carry it with them during the day. During this time the device broadcasts and receives information to and from other SPOT devices. When the user returns to their home computer that information can be viewed.

The goal of this project is to provide insight into what might be possible social networking was extended to mobile ad hoc networks.

Acknowledgments

I would like to express my appreciation to my advisor, Dr. Paul Gestwicki, who provided the idea for this project and for his oversight and input during its design. I would also like to express thanks to Dr. Fu-shing Sun who provided a great deal of input and support as well. Thanks are also in order to Sun Microsystems who provided the SunSPOT devices on a grant to Ball State University.
# Table Of Contents

1) Introduction 4

2) Implementation 6
   1. SPOT to SPOT communication 7
   2. SPOT to Host communication 8
   3. The GUI 9

3) Related Work 11

4) Conclusions 12
   1. What has been accomplished 12
   2. Future Work 12

5) Bibliography 14

6) Source Code  Appendix A
1. Introduction

Some of the most trafficked websites on the Internet today are social networking sites. In January of 2008 the social networking site Facebook.com reported 60 million registered users with an average of 250,000 new accounts being created daily since January 2007 (Facebook, Myspace Statistics, 2008). Social networking websites are online communities where a user can keep in contact with friends or acquaintances that have an account with the same service. One of the earliest social networking sites, Classmates.com, began in 1995 with the idea renewing ties with former school mates people had lost touch with. The world of electronic social networking has grown a great deal in recent years. The aforementioned Facebook.com was reported as the 6th most trafficked site on the Internet (Facebook, Myspace Statistics, 2008). Another major social networking site MySpace.com reports 110 million monthly active users around the world. In January 2008 the site reported a record 4.5 billion page views in one day (Facebook, Myspace Statistics, 2008). Many other sites exist for social networking that try to unite people through common interests or ties. Business contacts can stay in touch through LinkedIn, Advogato aims to unite free and open source software developers, Avatars United is a community for people to connect online game avatars and there are hundreds of others. Social networking is a phenomenon that continues to grow.

The popularity of wireless devices has skyrocketed in the last decade as well. According to Reuters, half the world population, 3.3 billion people, have cellular phones. Devices such as cellphones are now being made with Bluetooth, which supports ad hoc networking. Ad hoc networks differ from other wireless networks in that they are decentralized. Each node, or member, of an ad hoc network can communicate to every other node or has their information forwarded to a node it is not in direct contact with through other nodes. Other wireless networks have a central hub through which all communication is routed. Communication between members of a centrally managed network send their information to the central hub which then sends it to the desired destination.
The purpose of this research is to find out what might be possible if these two technological phenomena were combined. To answer this question the use of a device created by Sun Microsystems called the SunSPOT was used. SPOT stands for Small Programmable Object Technology. This device is perfectly suited for this kind of research as the devices are about the size of a small cellular phone, have built in protocols ad hoc networking, and allow for reprogramming using the Java programming language.
2. Implementation

My solution to creating a wireless social network using SunSPOTs has three main requirements. The first is that there must be communication, broadcasting and receiving of information, between SPOT devices. The transmitting of data between devices in an ad hoc network is the main driving idea behind this project.

SunSPOTs use the 802.15.4 wireless standard for ad hoc communication. This differs from the more commonly known 802.11 standard which is used for Wi-Fi communication. The 802.15.4 standard aims to provide a base for low-speed and low-cost ubiquitous computing. The framework of this standard assumes a ten meter communication area with a low transfer rate of 250 kilobits per second. In contrast the 802.11g standard, the common wireless standard for laptop computers provides for communication rates up to 54 megabits per second with a range up to 35 meters. The higher bandwidth and range is better for laptop computers because of the powerful processors and long battery life as compared to smaller devices such as the SunSPOT. The 802.15.4 standard is a good fit for the SunSPOT and other small wireless devices because the short range, low transfer rate and low power consumption.

The application protocol used on top of these standards was decided to be a datagram protocol. A datagram protocol sends out information in packets but does not request confirmation that each packet was received. This protocol works well for this project because data is broadcast indiscriminately to any destination that is listening.

The second is that the SPOT must be able to communicate, again broadcasting and receiving of data, to a host program running on a desktop computer. This requirement is necessary so that the user can communicate with their SPOT. The SPOT has no screen or simple user input device so a host computer is needed for the user to interact with the SPOT device.

The third requirement was that the host program should be simple and intuitive for users so that
interaction between the user and the SPOT is not complicated. This requirement is necessary because of the third requirement which makes the use of a computer necessary in communicating with the SPOT.

2.1 SPOT to SPOT Communication

This requirement is met by two different threads. While the SPOT does support non-blocking IO, I use two threads to keep the functions separate and to ensure continuous processing.

The first thread used to meet this requirement constantly broadcasts the information stored on the SPOT that is flagged for broadcast. This thread first opens up a radiogram connection on channel 11. The thread then runs a loop where it takes the information flagged for broadcasting that is stored in a string and writes it into a datagram. This string contains control characters so that it can be separated into individual messages when uploaded to the host. The datagram is then broadcast on the connection channel it opened earlier. At the end of each cycle of the loop it sleeps for 1000 milliseconds so that a receiving SPOT has time to receive the datagram and add the address of the sender to a list of addresses it has received broadcasts from. 1000 milliseconds may not be the ideal time for sleep as multiple transmissions do sometimes occur, but in testing was found to have an error occur in approximately one out of every twenty tries.

The second thread used to meet this requirement is a thread set to constantly listen for datagrams being broadcast. This thread begins like the other thread, opening up a radiogram connection on channel 11, the same channel as SPOTs broadcast on. The thread then runs a loop where it first listens to receive any datagram that is being broadcast. The loop then checks the address of the sender against its list of addresses of previously received datagrams. If the address does not appear on the list of previously received addresses the information is stored to memory and the addresses is added to the list.
2.2 SPOT to HOST Communication

This requirement is fulfilled by several threads on both the SPOT device and the host computer. The first thread on the SPOT constantly listens to the host for new information to broadcast. The thread opens a radiogram connection on channel 10 and adds the new information received from the host to the variable flagged for broadcast to other SPOTs. The host radiogram channel is set differently than the SPOT radiogram channel so that conceivably each SPOT could listen to a host on different channels and no unwanted communication between different hosts and SPOTs would occur.

The second thread on the SPOT to meet this requirement is set to continuously broadcast to the host. This thread opens a radiogram connection on channel 10 and transmits only the new information received from other SPOTs to the host. This thread could be changed to transmit only when a button is pressed, but as battery-life was not taken as a concern in this prototype it has been set to broadcast continuously.

The host side of the program also has to be able to communicate to the SPOT device and this is fulfilled by two functions, one to broadcast to the SPOT and one to listen to the SPOT. On the GUI for the host program there are two tabs, one with the broadcasting function and one with the listening function. The broadcast function takes input into a text field and when the “broadcast” button is pressed it sends out a datagram on channel 10, the same channel that the SPOT listens on for information from the host. The receiving function on the host runs when the “receive” button is pressed. This button runs a function which opens a radiogram connection on channel 10 and receives a datagram from the SPOT. The information received is then displayed in a text area for the user to read.
A very important part to any user interactive program is a graphical user interface. Since the SPOT and host programs were both written in Java, I decided to use Java's Swing toolkit which contains all the libraries need to make a GUI. The GUI is simple, it contains two tabbed panes, one with the broadcast functions and one with the receiving functions. On the “broadcasting” pane there is a text box to receive the user input, a button that when pressed runs the function to broadcast that user input, and a button to end the program. On the “receiving” pane there is a text area to display the received data, a button that when pressed receives the data from the SPOT and displays that data in the text area, a button to clear the text area, and a button to exit the program. This GUI is very simple but provides all the necessary functions for a minimal prototype like this project.

A production version of the GUI might contain more functionality. Things like URLs could be automatically loaded into a web browser when a certain button was pressed. A function to save all
data into a file and functions to change broadcast or listening channels could be added. To better
determine what functionality could be important, usability tests would be very helpful. User feedback
would shed new light on what would be handy and what features would make the product better.
3. Related Work

The ad hoc network that this project uses is possible due to the SunSPOT device. The SPOT device is also known as a Mote. Motes are devices that contain sensors and radio communications to be used in mobile sensor networks. Typical applications for motes use an ad hoc network to communicate sensor information over a large area or in an area where wires connecting to a central computer would not be possible. A situation where motes might be helpful could be in water or power metering. Motes could be attached to the meters in a neighborhood and information could be transmitted to a truck that drives by instead of a person having to individually check each meter.

A unique feature of the SunSPOT device is its ability to have a program automatically uploaded on to it. Many motes use an operating system, notably TinyOS. This is an embedded operating system used for wireless sensor networks. Programs for TinyOS are written in the nesC programming language which is similar to the C programming language, but is optimized for the limited memory provided by motes. Programs are uploaded to the mote and then run in TinyOS which provides all of the interfaces to the components of the mote. The SunSPOT uses a different approach to uploading and running a program on the mobile device. The first difference is that programs are written in Java micro edition. Java micro edition is basically a stripped down set of Java standard edition that is made for mobile devices. Certain libraries and functions are not in JavaME so that the processor and memory capabilities of smaller devices can run the program easily. The second difference is the way in which programs are uploaded and run on the device. To upload a program to the SPOT, a programmer runs a deploy script on their workstation computer which compiles and sends the program to the spot. The program is then run on the SPOT by the Squawk Virtual Machine. The SquawkVM differs from TinyOS in that it is not a fully fledged operating system. The SquawkVM provides only what is needed to run programs written in JavaME and therefore provides for much less overhead than a application written for TinyOS.
4. Conclusions

4.1 What has been accomplished

Only a limited prototype of this research has been created. Use of the system begins with the host program with the GUI. The GUI is very minimal with the upload pane only having an area to type information and a button to submit that information to upload to the SPOT. The download pane is also minimal with an area to view the information stored on the SPOT, a button to start the download process and a button to clear the area. The SPOT to host and host to SPOT communication process works well using over the air transmission from the base station connected to the host.

The SPOT to SPOT communication process works well, but as previously mentioned there is a bug in the function to check for previously received addresses. Sometimes repeat broadcasts are received before the address is blocked. The information storage process on the SPOT is quite simple and efficient using a single string with control characters to separate the information.

This project is somewhat of a proof-of-concept. It shows how a wireless social network might work and interact. More importantly however, it springs many ideas of what might be possible if this technology became more widespread.

4.2 Future work

There are many possible ways in which this work could continue. As previously mentioned more GUI work could be done to add functionality to the program. Also previously mentioned is the bug in the listening function where repeat broadcasts are received. Besides continuing work on the prototype network on SPOTs, the program could be reworked to run on ad-hoc network capable cellphones, a much more popular device. The aim of this project was to see what might be possible in extending a social network to wireless devices. The SPOT allows for a proof of concept, but does not allow much room for user interaction with the device itself. If the project were extended to cellphones
or PDAs, an entirely new set of problems would arise. Functionality for user interaction on the wireless device itself would have to be added. Porting this project to cellphones would also make usability testing much easier as a wider audience could be reached since the specialized SunSPOT device would not be needed.

An extension of this project to other more commonly carried devices could be quite useful. Studies could be run to see how many people cross paths in a given day. This could be helpful in studies on the spread of disease. Useful advertising schemes could also be set up so that you get information about events or specials about places you walk by or go to. A person could be notified of an upcoming special at a restaurant they are currently eating in, or notified of upcoming performances when walking by a venue on the street.
Bibliography


package org.sunspotworld;

//Packages for Spot Radio communication
import com.sun.spot.peripheral.Spot;
import com.sun.spot.peripheral.radio.IRadioPolicyManager;
import com.sun.spot.io.j2me.radiostream.*;
import com.sun.spot.io.j2me.radiogram.*;
import com.sun.spot.util.IEEEAddress;

//Packages for GUI
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import javax.microedition.io.*;

/**
 * Host application
 */
public class SunSpotHostApplication{

    //List of components used in building the GUI
    JFrame jtfMainFrame;
    JButton jbnButton1, jbnButton2, jbnButton3, jbnButton4, jbnButton5;
    JTextField jtfInput;
    JPanel jplPanel, jplPanel2;
    JTextArea jAreaOutput;
    JScrollPane jsPane;
    String send = "";

    //Run function - where all the actual work is done.
    public void run() {

        JTabbedPane jtbPane = new JTabbedPane();

        //Panel 1 Components

        //Input text field
        jtfInput = new JTextField(40);

        //Button to transfer information in text field to SPOT
        jbnButton1 = new JButton("Transfer To Spot");
        jbnButton1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

                try {
                    DatagramConnection sendConn = (DatagramConnection)
                                              Connector.open("radiogram://broadcast:10");

                    Datagram dg = sendConn.newDatagram(sendConn.getMaximumLength());
                    dg.writeUTF(send);
                    sendConn.send(dg);
                } catch (IOException ex) {
                    ex.printStackTrace(System.out);
                } catch (ClassNotFoundException cnfe) {
                    cnfe.printStackTrace(System.out);
                } catch (Exception ex) {
                    ex.printStackTrace(System.out);
                }
            }
        });

        jtbPane.addTab("Panel 1", null, jtfInput, null);
        jtbPane.addTab("Panel 2", null, jAreaOutput, null);

        jtfMainFrame = new JFrame("JFrame Title");
        jtfMainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jtfMainFrame.add(jtbPane);
        jtfMainFrame.setSize(500, 750);
        jtfMainFrame.setVisible(true);
    }

}
sendConn.close();
jtInput.setText(" ");
}
catch(Exception ex)
{
    ex.printStackTrace();
}

// Exits Program
jbtnButton2 = new JButton("Exit");
jbtnButton2.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});

// Panel 2 Components

// Button to receive information stored on SPOT
jbtnButton3 = new JButton("Transfer From Spot");
jbtnButton3.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {
        DatagramConnection recvSpotConn;
        try {
            recvSpotConn = (DatagramConnection) Connector.open("radio//:12");
            Datagram dg = recvSpotConn.newDatagram(recvSpotConn.getMaximumLength());
            recvSpotConn.receive(dg);
            String received = dg.readUTF();
            recvSpotConn.close();
            jtAreaOutput.setText(" ");
            jtAreaOutput.append(received + "\n");
        } catch (IOException ex) {
            ex.printStackTrace();
    }
});

// Button to exit program
jbtnButton4 = new JButton("Exit");
jbtnButton4.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});

// Button to clear field with output from SPOT
jbtnButton5 = new JButton("Clear");
jbtnButton5.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e) {
        jtAreaOutput.setText(" ");
    }
});

jtAreaOutput = new JTextArea(10, 40);
jtAreaOutput.setEditable(false);
jsPane = new JScrollPane(jtAreaOutput);

// JPanel - Add Components
jplPanel = new JPanel();
jplPanel.setLayout(new FlowLayout());
jPanel.add(jtfInput);
jPanel.add(jbtnButton1);
jPanel.add(jbtnButton2);

jPanel2 = new JPanel();
jPanel2.setLayout(new FlowLayout());
jPanel2.add(jsPane);
jPanel2.add(jbtnButton5);
jPanel2.add(jbtnButton3);
jPanel2.add(jbtnButton4);

jOptionPane.addTab("Transfer To Spot", jPanel);
jOptionPane.addTab("Transfer From Spot", jPanel2);

// JFrame Components
jtfMainFrame = new JFrame("Wireless Social Network Host Application");
jtfMainFrame.getContentPane().add(jOptionPane, BorderLayout.CENTER);
jtfMainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
jtfMainFrame.pack();
jtfMainFrame.setVisible(true);
}

// Main Function
public static void main(String[] args) {
   SunSpotHostApplication app = new SunSpotHostApplication();
   app.run();
}
}
package org.sunspotworld;

//Packages with drivers for sensorboard and radio communication
import com.sun.spot.peripheral.IFlashMemoryDevice;
import com.sun.spot.peripheral.Spot;
import com.sun.spot.sensorboard.EDemoBoard;
import com.sun.spot.sensorboard.peripheral.ISwitch;
import com.sun.spot.sensorboard.peripheral.ITriColorLED;
import com.sun.spot.peripheral.radio.IRadioPolicyManager;
import com.sun.spot.io.j2me.radiostream.*;
import com.sun.spot.io.j2me.radiogram.*;
import com.sun.spot.util.*;
import com.sun.squawk.imp.MIDletMainWrapper;
import java.io.*;
import javax.microedition.io.*;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class StartApplication extends MIDlet {

    //Initialization of variables
    String toBroadcast = " ";
    String received = " ";
    int count = 0;
    String[] addresses = new String[100];

    //This function contains 4 threads. Two for listening and broadcasting to the host and two for listening
    //broadcasting to other spots. The reason there are different threads for host and SPOT is so that
    //SPOT could conceivably have a different channel for their unique host so that interference between
    //the host and other SPOTs would not occur.

    //This thread constantly listens on channel 10, the channel that the host basestation broadcasts on
    public void ListenToHostThread() throws IOException {
        new Thread() {
            DatagramConnection recvHostConn = (DatagramConnection)
                    Connector.open("radiogram://:10");
            Datagram dg = recvHostConn.newDatagram(recvHostConn.getMaximumLength());

            public void run(){
                while(true){
                    try{
                        recvHostConn.receive(dg);
                        toBroadcast = toBroadcast + "\n" + dg.readUTF();
                    } catch (IOException ex) {
                        ex.printStackTrace();
                    }
                }
            }.start();
        }
    }

    //This channel listens on channel 11, the channel that all SPOTs broadcast to other SPOTs on
    public void ListenToSPOTSThread() throws IOException {
        new Thread() {
            DatagramConnection recvSpotConn = (DatagramConnection)
                    Connector.open("radiogram://:11");
            Datagram dg = recvSpotConn.newDatagram(recvSpotConn.getMaximumLength());
            String newaddress = " ";
            public void run() {
                while(true){
                    try{
                        recvSpotConn.receive(dg);
                        newaddress = newaddress + "\n" + dg.readUTF();
                    } catch (IOException ex) {
                        ex.printStackTrace();
                    }
                }
            }.start();
        }
    }
}
```java
try {
    recvSpotConn.receive(dg);
    newaddress = dg.getAddress();
    if (addressCheck(newaddress) == true) {
        received = received + "\n" + dg.readUTF() + " From: " + newaddress;
        addresses[count] = newaddress;
        count++;
    }
} catch (IOException ex) {
    ex.printStackTrace();
}
}.start();

//This thread broadcasts on channel 12, the channel that the Host listens on
public void BroadcastToHostThread() throws IOException {
    new Thread() {
        DatagramConnection sendConn = (DatagramConnection)
            Connector.open("radiogram://broadcast:12");
        Datagram dg = sendConn.newDatagram(sendConn.getMaximumLength());
        public void run() {
            while (true) {
                try{
                    dg.reset();
                    dg.writeUTF(toBroadcast + "\n" + received);
                    sendConn.send(dg);
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
                Utils.sleep(1000);
            }
        }.start();
    }
}

//This thread broadcasts on channel 11, the channel that all SPOTs listen on for
communication to other SPOTs
public void BroadcastToSPOTsThread() throws IOException {
    new Thread() {
        DatagramConnection sendConn = (DatagramConnection)
            Connector.open("radiogram://broadcast:11");
        Datagram dg = sendConn.newDatagram(sendConn.getMaximumLength());
        public void run() {
            while (true) {
                try{
                    dg.reset();
                    dg.writeUTF(toBroadcast);
                    sendConn.send(dg);
                } catch (IOException ex) {
                    ex.printStackTrace();
                }
                Utils.sleep(1000);
            }
        }.start();
    }
}

//Function to check for duplicate addresses
public boolean addressCheck(String address)
{  
    int i=0;
    while (i<=count ){
        if(addresses[i] == address)
            return false;
        i++;
    }
    return true;
}
protected void startApp() throws MIDletStateChangeException {
    try {
        ListenToHostThread();
        ListenToSPOTsThread();
        BroadcastToHostThread();
        BroadcastToSPOTsThread();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
}
protected void pauseApp() {
    // This is not currently called by the Squawk VM
}
protected void destroyApp(boolean unconditional) throws MIDletStateChangeException {
    }
}