DATA COLLECTION IN WIRELESS SENSOR NETWORKS USING UAV MOBILE SINK

Noushin KARIMPOUR

Contents

1. IRIS Motes and Parrot Drone

In this project I have used IRIS nodes as sensor, mobile and base station nodes and Parrot Drone to create a mobile node. The Parrot Drone has a mobile app that allows to user to control the Drone. The IRIS motes support TinyOS operating system and nesC language. Figure 1 shows the Parrot Drone, IRIS mote and an MDA100 sensor board.

Figure 1. A) Parrot Drone B) IRIS mote, C) MDA100 sensor

2. Base Station

I uploaded the TinyOS base station program to one of the motes and attached it to computer to get the packets in Java application. The base station program reads the packets from radio and forwards them to serial ports. The sent packets to serial port are read by a java application which is explained in next sections.

3. Sensor Node

The sensor node reads the temperature data from MDA100 sensor board and broadcast them to all other nodes in its radio range. I set the radio range of nodes to 1.5 m. In the boot function I starts the radio controller. Every time that the Timer fires I read the temperature data from sensor board.

```
event void Boot.booted() {
call AMControl.start();
}

event void Timer.fired()
{
  call Read.read();
}
```

If the radio starts without error I start the timer periodically for every 1 seconds. Otherwise I try to start the radio controller again.

```
event void AMControl.startDone(error_t err) {
  if (err == SUCCESS) {
    call Timer.startPeriodic(SAMPLING_FREQUENCY);
  }
  else {
   call AMControl.start();
  }
}
```

When the temperature data receives from sensor I create a packet and put the data into the packet and then broadcast the packet over radio. The following function shows the data read and broadcast codes.

```
event void Read.readDone(error_t result, uint16_t data)
{
  call Leds.led0Toggle();
  if (result == SUCCESS){
   radio_msg_t* rcm = (radio_msg_t*)call Packet.getPayload(&packet, sizeof(radio_msg_t));
    if (rcm == NULL) {return;}
    rcm->data = data;
    if (call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(radio_msg_t)) == SUCCESS) {
    call Leds.led1Toggle();
    }
  }
}
```

4. Mobile Node

The mobile node wait for incoming messages and rebroadcast the last received message periodically every 1 second. The received data is stored in the received_data variable which its initial value is 0.

```
message_t packet;
uint16_t received_data;
event void Boot.booted() {
call AMControl.start();
received_data=0;
}
```

When the radio controller of the mobile node started I start a timer periodically to rebroadcast the incoming message from other nodes.

```
event void AMControl.startDone(error_t err) {
  if (err == SUCCESS) {
    call Timer.startPeriodic(1000);
  }
  else {
   call AMControl.start();
  }
}
```

In the timer function of the mobile node if the received_data is not 0 then the node creats a packet and broadcast the received_data value. In this way the last received data by the mobile node is broadcasted to all other nodes in the radio range of node. The following function shows the timer function of mobile node.

```
event void Timer.fired()
{
    radio_msg_t* rcm = (radio_msg_t*)call Packet.getPayload(&packet, sizeof(radio_msg_t));
    if (received_data==0) return;
    if (rcm == NULL) {return;}
    rcm->data = received_data;
    if (call AMSend.send(AM_BROADCAST_ADDR, &packet, sizeof(radio_msg_t)) == SUCCESS) {
    call Leds.led1Toggle();
    }
    call Leds.led0Off();
}
```

When the mobile node receive a data from another node it store the received value in the received_data variable and also turn on the leds. In this way in the next timer fire it will rebroadcast the incoming message.

```
event message_t* Receive.receive(message_t* bufPtr, void* payload, uint8_t len) {
  radio_msg_t* rcm = (radio_msg_t*)payload;
  call Leds.led2On();
  call Leds.led0On();
  received_data=rcm->data;
}
```

5. Java Application

In the Java application the received packets from base station are displayed in a GUI. Figure 2 shows the GUI of java program.

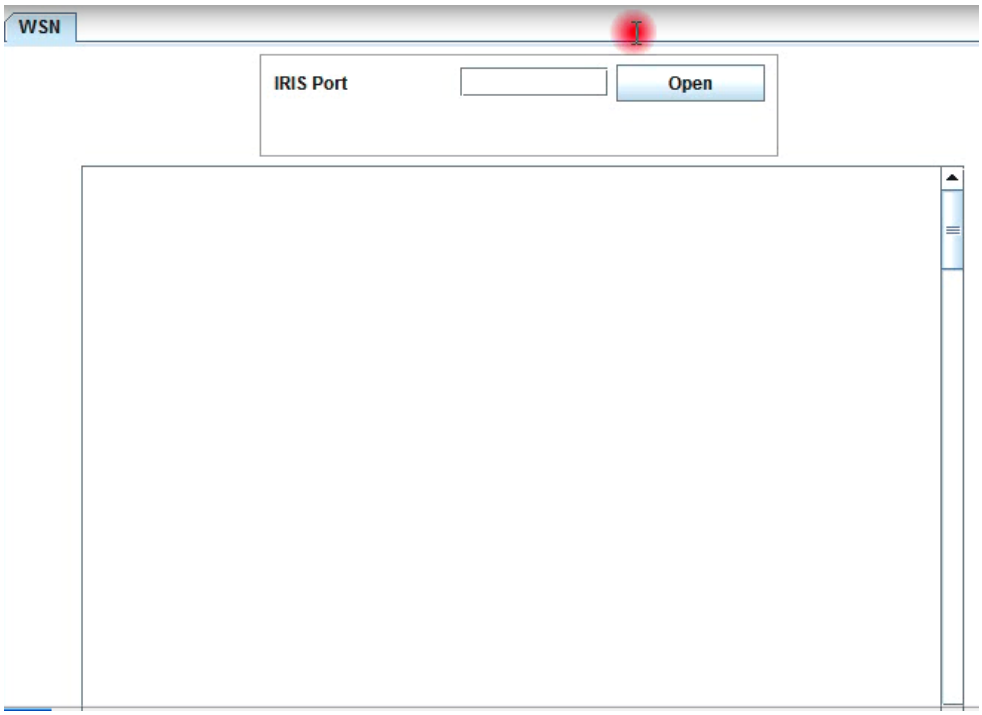

Figure 2. GUI of the java program.

The main class of the Java program is PortListener class which opens the serial port and forward the incoming packets to the window class. The following function shows the port opening codes. This function is called when the user click on the open button. In this function we create a reader object and assign it to the specified port number.

```
public PortListener(Frame1 frm, String[] args) {
    window = frm;
    try {
      if (args.length == 2 && args[0].equals("-comm")) {
        source = args[1];
      } else if (args.length > 0) {
        System.err.println("usage: java net.tinyos.tools.Listen [-comm PACKETSOURCE]");
       System.exit(2);
      }
      if (source == null) {
        reader = BuildSource.makePacketSource();
      } else {
        reader = BuildSource.makePacketSource(source);
      }
      if (reader == null) {
        System.err.println("Invalid packet source (check your MOTECOM environment variable)");
        System.exit(2);
      }
      reader.open(PrintStreamMessenger.err);
```

```
            window.display("Port Opened. Waiting for incoming packets..");
            STime = Calendar.getInstance().getTimeInMillis();
        } catch (Exception ex) {
            window.display(ex.toString());
        }
    }
```

After opening the port, in the following function we wait for incoming bytes from the port and convert the received bytes to a packet object and deliver the packet to the window object which displays in the screen.

```
public void run() {
        mt = new MyTask(this);
        Timer timer = new Timer();
        timer.schedule(mt, 1000);
        try {
            Packet p = new Packet();
            while (true) {
                byte[] packet = reader.readPacket();
                p = generatePacket(packet);
                window.display("[RECV]  " + bytesToHex(packet)+"   ("+p.data+")");
            }
        } catch (Exception e) {
            window.display(e.toString());
        }
    }
```