Lab 4 Mote-PC Serial Communication

Due on Mar. 2nd

This lab is to implement mote-PC communication through serial port. The principle is almost the same as the mote-mote communication. The only difference is that it is using serial port instead of radio.

1. The first step testing serial port is to install the `apps/tests/TestSerial` application on a mote. This application sends a packet to the serial port every second.

Once you have installed `TestSerial`, you need to run the corresponding Java application that communicates with it over the serial port. This is built when you build the TinyOS application. From in the application directory, type:

```
$ java TestSerial
```

If it is not working, try add a parameter –comm, type:

```
$ java net.tinyos.tools.Listen -comm serial@/dev/ttyUSB1:iris
```

tells Listen to use the serial port#dev/ttyUSB1#at the correct speed for a iris mote.

If you run#TestSerial#with the proper PORT and SPEED settings, you should see output like this:

```
Sending packet 1
Received packet sequence number 4
Sending packet 2
Received packet sequence number 5
Sending packet 3
Received packet sequence number 6
Sending packet 4
Received packet sequence number 7
Received packet sequence number 8
Sending packet 5
Received packet sequence number 9
Sending packet 6
```

If you do not want to add the parameter –comm every time when communicate with serial port, you can set MOTECOM at Tinyos.sh file.

```
export MOTECOM=serial@/dev/ttyUSB1:iris
```

2. BaseStation is a basic TinyOS utility application. It acts as a bridge between the serial port and radio network. When it receives a packet from the serial port, it transmits it on the radio; when it receives a packets over the radio, it transmits it to the serial port. Because TinyOS has a toolchain for generating and sending packets to a mote over a serial port, using a BaseStation allows PC tools to communicate directly with mote networks.

Take the node that can send data through radio from Lab3 and install BaseStation on another node. Suppose the application that can send data is called BlinkToRadio. If you turn on the node that still has BlinkToRadio installed, you should see LED 1 on the BaseStation blinking. BaseStation toggles LED 0 whenever it sends a packet to the radio and LED 1 whenever it sends a packet to the serial port. It toggles LED 2 whenever it has to drop a packet: this can happen when one of the two receives packets faster than the other can send them.

BaseStation is receiving your BlinkToRadio packets and sending them to the serial port, so if it is plugged into a PC we can view these packets. The Java tool Listen is a basic packet sniffer: it prints out the binary contents of any packet it hears. Run Listen, using either MOTECOM or a -comm parameter:

```
$ java net.tinyos.tools.Listen
```

Listen creates a packet source and just prints out every packet it sees. Your output should look something like this:

```
00 FF FF 00 00 04 22 06 00 02 00 01
00 FF FF 00 00 04 22 06 00 02 00 02
00 FF FF 00 00 04 22 06 00 02 00 03
00 FF FF 00 00 04 22 06 00 02 00 04
00 FF FF 00 00 04 22 06 00 02 00 05
00 FF FF 00 00 04 22 06 00 02 00 06
00 FF FF 00 00 04 22 06 00 02 00 07
00 FF FF 00 00 04 22 06 00 02 00 08
00 FF FF 00 00 04 22 06 00 02 00 09
00 FF FF 00 00 04 22 06 00 02 00 0A
00 FF FF 00 00 04 22 06 00 02 00 0B
```

3. The Listen program is the most basic way of communicating with the mote; it just prints binary packets to the screen. Obviously it is not easy to visualize the sensor data using this program. What we'd really like is a better way of retrieving and observing data coming from the sensor network. Of course, exactly what data to display and how to visualize it can be very application specific. For this reason, TinyOS only has a few applications for visualizing simple sensor , but it provides support for building new visualization or logging systems.

Let's build a Java packet object for BlinkToRadio. Before building, we can look at the makefile of TestSerial as a reference:

```
COMPONENT=TestSerialAppC
BUILD_EXTRA_DEPS += TestSerial.class
CLEAN_EXTRA = *.class TestSerialMsg.java

TestSerial.class: $(wildcard *.java) TestSerialMsg.java
        javac *.java

TestSerialMsg.java:
        mig java -target=null -java-classname=TestSerialMsg TestSerial.h TestSerialMsg -o $@

include $(MAKERULES)
```

Now we can build java packet object for BlinkToRadio. Open the Makefile for BlinkToRadio and add a dependency:

```
BUILD_EXTRA_DEPS=BlinkToRadioMsg.class
```

Then add a step which explains how to compile a .java to a .class:

```
BlinkToRadioMsg.class: BlinkToRadioMsg.java
        javac BlinkToRadioMsg.java
```

Note that there must be a tab before javac, and not just spaces. Finally, add the line which explains how to create BlinkToRadioMsg.java:

```
BlinkToRadioMsg.java:
        mig java -target=null -java-classname=BlinkToRadioMsg BlinkToRadio.h BlinkToRadioMsg -o
$@
```

As with javac, there must be a tab (not spaces) before mig. Now, when you type make in BlinkToRadio/, the make system will compile BlinkToRadioMsg.class, a Java class that parses a binary packet into message fields that can be accessed through methods.

```
warning: Cannot determine AM type for BlinkToRadioMsg
         (Looking for definition of AM_BLINKTORADIOMSG)
```

One part of the TinyOS communication toolchain requires being able to figure out which AM types correspond to what kinds of packets. To determine this, for a packet type named X, mig looks for a constant of the form AM_X. The warning is because we defined our AM type as AM_BLINKTORADIO, but mig wants AM_BLINKTORADIOMSG. Modify BlinkToRadio.h so that it defines the latter. You'll also need to update BlinkToRadioAppC.nc so that the arguments to AMSenderC and AMReceiverC use it. Recompile the application, and you should see no warning. Install it on a mote.

Now that we have a Java message class, we can use it to print out the messages we see from the BaseStation. With BaseStation plugged into the serial port and BlinkToRadio running on another mote, from the BlinkToRadio directory type

```
java net.tinyos.tools.MsgReader BlinkToRadioMsg
```

Now, when the BaseStation sends a packet to the serial port, MsgReader reads it, looks at its AM type, and if it matches the AM type of one of the Java message classes passed at the command line, it prints out the packet. You should see output like this:

```
1152232617609: Message
  [nodeid=0x2]
  [counter=0x1049]

1152232617609: Message
  [nodeid=0x2]
  [counter=0x104a]

1152232617609: Message
  [nodeid=0x2]
  [counter=0x104b]

1152232617621: Message
  [nodeid=0x2]
  [counter=0x104c]
```

4. One problem with directly using the serial port is that only one PC program can interact with the mote. Additionally, it requires you to run the application on the PC which is physically connected to the mote. The SerialForwarder tool is a simple way to remove both of these limitations.

Most generally, the SerialForwarder program opens a packet source and lets many applications connect to it over a TCP/IP stream in order to use that source. For example, you can run a SerialForwarder whose packet source is the serial port; instead of connecting to the serial port directly, applications connect to the SerialForwarder, which acts as a proxy to read and write packets. Since applications connect to SerialForwarder over TCP/IP, applications can connect over the Internet.

SerialForwarder is the second kind of packet source. A SerialForwarder source has this syntax:

```
sf@HOST:PORT
```

HOST and PORT are optional: they default to localhost (the local machine) and 9002. For example,

```
sf@dark.cs.berkeley.edu:1948
```

will connect to a SerialForwarder running on the computer dark.cs.berkeley.edu and port 1948.

The first step is to run a SerialForwarder; since it takes one packet source and exports it as an sf source, it takes a packet source parameter just like the other tools we've used so far: you can pass a -comm parameter, use MOTECOM, or just rely on the default. Close your MsgReader application so that it no longer uses the serial port, and run a SerialForwarder:

```
java net.tinyos.sf.SerialForwarder
```

Since SerialForwarder takes any packet source as its source, you can even string SerialForwaders along:

```
java net.tinyos.sf.SerialForwarder -port 9003 -comm sf@localhost:9002
```

This command opens a second SerialForwarder, whose source is the first SerialForwarder. You'll see that the client count of the first one has increased to one. It's rare that you'd ever want to do this, but it demonstrates that in the message support libraries you can use a variety of packet sources.

Close the second SerialForwarder (the one listening on port 9003). Run MsgReader again, but this time tell it to connect to your SerialForwarder:

```
java net.tinyos.tools.MsgReader -comm sf@localhost:9002 BlinkToRadioMsg
```

You will see the client count increment, and MsgReader will start printing out packets.