CSC2026 Computer Networks

Mote-PC Communication

Aims:

This lesson is a continuation of last week's tutorial, introducing radio communications in TinyOS. If you have not completed last week's practical then please do so, and open last week's project. Today you will become familar with additional TinyOS interfaces and components that support communications, and you will learn how to:

- **Send a message buffer from mote to PC**

TASKS to help you achieve the above objectives are clearly labeled.

Mote-PC serial communication

This section presents the Java-based infrastructure for communicating with motes. It will allow you to send messages from your mote to the PC for debugging purposes.

Sending messages to the PC

TASKs:

1. Modify BlinkToRadioAppC.nc to declare the components for serial communication and connect their interfaces to your application:

```
components SerialActiveMessageC; //new
components new SerialAMSenderC(AM_BLINKTORADIO); //new

App.SerialControl -> SerialActiveMessageC; //new
App.SerialSend -> SerialAMSenderC; //new
```

2. Modify BlinkToRadioC.nc to declare the interfaces for serial communication:

```
uses interface SplitControl as SerialControl; //new
uses interface AMSend as SerialSend; //new
```

3. Define the required event handlers for the new interfaces:

```
event void SerialControl.startDone(error_t err) {
 if (err == SUCCESS) {
  } else {
    call SerialControl.start();
  }
}
event void SerialControl.stopDone(error_t err) {}
event void SerialSend.sendDone(message_t* msg, error_t error) {}
```

4. Initialise the serial interface component:

```
  event void Boot.booted() {

  call AMControl.start();

  call SerialControl.start(); //new

}
```

5. Send a message to the PC after it is received from Echo:

```
event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len) {
   if ((len == sizeof(BlinkToRadioMsg)) && (call AMPacket.destination(msg) ==
TOS_NODE_ID)) {
     BlinkToRadioMsg* btrpkt = (BlinkToRadioMsg*)payload;
     call SerialSend.send(TOS_NODE_ID, msg, sizeof(BlinkToRadioMsg)); // new
     call Leds.set(btrpkt->counter);
   }
   return msg;
 }
```

6. Now compile your application and install it on the mote. Don't forget to append your unique number:

```
 make gnode install,number
```

7. If your application is working and there is an Echo mote within radio range, your leds should be counting as before.

Displaying mote messages on your PC

TASKs (Sign of with Demonstrator when finished):

1. Your application on the mote should now be sending messages to the PC but you cannot yet see them. From within the Terminal window you used to install your mote application, type the following command. Note that on some keyboards, pressing the @ key may produce the " symbol. In this case, pressing the " key should produce the @ symbol:

```
Within your VM Navigate to the Java directory:

 cd /home/tinyos/tinyos-2.x/support/sdk/java
```

   Type in the following command:

```
java -cp tinyos.jar net.tinyos.tools.Listen –comm  serial@/dev/ttyUSB0:57600
```

2. After a short delay, you should see a series of lines of hexadecimal numbers, similar to those shown below. Each line represents one message. The messages show the return message from Echo.

- 00 00 0C 00 00 02 01 06 00 0C 00 0B
- 00 00 0C 00 00 02 01 06 00 0C 00 0C
- 00 00 0C 00 00 02 01 06 00 0C 00 0D
- 00 00 0C 00 00 02 01 06 00 0C 00 0E
- 00 00 0C 00 00 02 01 06 00 0C 00 0F
- 00 00 0C 00 00 02 01 06 00 0C 00 10
- 00 00 0C 00 00 02 01 06 00 0C 00 11

The first byte is always zero (00).

Two bytes represent the source node for the message, most significant byte first (00 0C). This should be the HEX representation of your unique ID.

Two bytes represent the destination node for the message, most significant byte first (00 02). (Echo mote ID)

1 byte represents the AM message number defined in the BlinkToRadio.h header file (06).

The last 4 bytes represent the payload. In this case btrpkt->nodeid (00 0C) and btrpkt->counter (00 0B).

You can terminate the Listen program with Ctrl-C.

3. Unless Echo is exceptionally busy, every message sent will be echoed. This would of course not be a very good test of a protocol designed to cope with missing messages. So far, you have being sending all messages on AM channel 6 (see BlinkToRadio.h). Change the value of AM_BLINKTORADIO to 99. Recompile and install your application. Run the java Listen program again and observe the output. You should see that Echo drops about one message in every four/five. In the example below, messages 1B and 22 are sent to Echo but not echoed.

```
00 00 0C 00 00 02 01 63 00 0C 00 16
00 00 0C 00 00 02 01 63 00 0C 00 17
00 00 0C 00 00 02 01 63 00 0C 00 18
00 00 0C 00 00 02 01 63 00 0C 00 19
00 00 0C 00 00 02 01 63 00 0C 00 1A
00 00 0C 00 00 02 01 63 00 0C 00 1C
00 00 0C 00 00 02 01 63 00 0C 00 1D
00 00 0C 00 00 02 01 63 00 0C 00 1E
00 00 0C 00 00 02 01 63 00 0C 00 1F
00 00 0C 00 00 02 01 63 00 0C 00 20
00 00 0C 00 00 02 01 63 00 0C 00 21
00 00 0C 00 00 02 01 63 00 0C 00 23
00 00 0C 00 00 02 01 63 00 0C 00 24
00 00 0C 00 00 02 01 63 00 0C 00 25
00 00 0C 00 00 02 01 63 00 0C 00 26
```

4. Change AM_BLINKTORADIO back to 6.