

# Sensor Network Lab Exercises Using TinyOS and MicaZ Motes, Part II

Jens Mache, Elgin Dean, and Kevin Imber

Department of Mathematical Sciences  
Lewis & Clark College, Portland OR, USA  
jmache@lclark.edu

**Abstract** *MIT Technology Review lists sensor networks as one of “Ten Emerging Technologies That Will Change the World” [1]. This paper extends our earlier paper [7], and describes three additional lab exercises, derived from the TinyOS tutorials and using MicaZ motes, that are suitable for activity-driven teaching of sensor networks to undergraduate students. One exercise includes some actual coding, and in another exercise, a mote also logs data to local storage, and reports it back to the PC at a later time. We describe our experiences, lessons and code alterations.*

**Keywords:** Wireless sensor networks, Education, TinyOS, MicaZ motes, Pervasive computing.

## 1 Introduction

Sensor networks are on their way to becoming ubiquitous in our everyday lives. In the future, agricultural fields may be fitted with sensor network devices letting farmers know when to water fields, nursing homes could use sensor networks to alert nurses of possible problems [4], and eventually, sensor network devices may even be embedded in concrete structures such as bridges to monitor structural integrity. With such a large potential for applications, it is surprising that only a few colleges and universities have introduced sensor networks into the undergraduate curriculum to date. It is critical that the next generation of computer scientists know how to program and deploy sensor networks. Although some tutorials currently exist, their target audience is largely graduate students and professional researchers.

This paper extends our earlier paper [7], in which we describe three introductory lab exercises that are suitable for the activity-driven teaching of sensor networks to undergraduate students. The first exercise was about uploading a simple program that blinks one LED to one mote, which is equivalent to

a *hello world* program. The second exercise used radio communication between several motes. The third exercise used a sensor board to take measurements and send them to a base station PC, which could then make them available over the Internet. In this paper, we describe more exercises and ideas for undergraduate sensor network education. Two undergraduate students and one faculty member spent a week working through the original TinyOS tutorials [2] to adapt these exercises.

## 2 Background

### 2.1 Hardware

Sensor network devices, called motes, are manufactured by companies such as Intel, Crossbow, Dust Networks, Millennial Net, Arch Rock, Ember, Sun Microsystems, and others. We had access to Crossbow MicaZ motes, including Mica sensor boards and MIB510 programming boards. For detailed hardware specifications, see the Crossbow website [5].

### 2.2 TinyOS

“TinyOS is an open-source operating system designed for wireless embedded sensor networks. It features a component-based architecture which enables rapid innovation and implementation while minimizing code size as required by the severe memory constraints inherent in sensor networks. TinyOS’s component library includes network protocols, distributed services, sensor drivers, and data acquisition tools—all of which can be used as-is or be further refined for a custom application. TinyOS’s event-driven execution model enables fine-grained power management yet allows the scheduling flexibility made necessary by the unpredictable nature of wireless communication and

physical world interfaces.” [3]

## 2.3 Setup

After several unsuccessful attempts to install TinyOS 2.x, both on Linux and Windows/Cygwin, we had success with the Windows InstallShield for version 1.1.11, which we still recommend at this time. Two things to note before getting started:

1. In this paper, paths are relative to the TinyOS directory, which by default is `/opt/tinyos-1.x`.
2. Do not attach a mote with batteries in it to a powered programming board.

## 3 Hands-on Exercises

### 3.1 Lab Exercise IV: Control

In this exercise, based on lesson 7 from the TinyOS 1.x tutorial [2], the PC wirelessly controls a mote by sending it packets that command it to do specific functions. Figure 1 shows which piece of code runs on each piece of hardware. The steps are as follows:

1. Connect the programming board to the serial port of the PC, attach a mote to the programming board, and connect the programming board to power.
2. Compile and upload the `SimpleCmd` application to the mote:
  - (a) `cd` into `apps/SimpleCmd`
  - (b) type `make <mote type> install <board type>,<device>`. In our case, the correct command was `make micaz install mib510,/dev/ttyS0`.
3. Remove the mote from the programming board and install batteries.
4. Attach another mote to the programming board.
5. Compile and upload the `TOSbase` application to this second mote.
6. Start the `SerialForwarder` application:

```
java net/tinyos.sf.SerialForwarder
-comm serial@<serialport>:<baud rate>
```

In our case, the command was:

```
java net/tinyos.sf.SerialForwarder
-comm serial@COM1:57600
```

7. In a new terminal window, use the `BcastInject` application to send a command to the mote running `SimpleCmd`:

```
java net/tinyos.tools.BcastInject
<command>
```

The supported commands are:

- `led_on` – Turn on the yellow LED
- `led_off` – Turn off the yellow LED
- `radio_louder` – Increase radio power
- `radio_quieter` – Decrease radio power

You should now be able to control the yellow LED and the radio power by running `BcastInject`. Try adjusting the radio power to see how it affects the range at which the mote can receive commands.

#### 3.1.1 Problems Encountered

The TinyOS tutorials imply that the baud rate can be left out or replaced by the model of the mote, but we found that we could only get `SerialForwarder` to work if we specified the correct baud rate for the mote when launching it (see step 6). The baud rate for Mica2 and MicaZ motes is 57600.

### 3.2 Lab Exercise V: Sounder Control

In this exercise, we will take the code from the previous exercise and modify it, adding a function to control the sounder built into the MicaZ’s sensor board. This exercise is based on the suggestion given in lesson 7 of the TinyOS tutorial. We recommend trying this on your own before reading the solution below. The steps are:

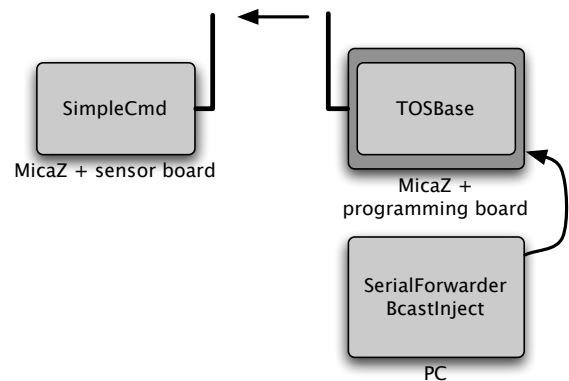


Figure 1: Setup and flow of data in exercises four and five.

1. Connect the programming board to the serial port of the PC, attach a mote and a sensor board to the programming board (sensor board goes below), and connect the programming board to power.
2. Make necessary changes to `SimpleCmd` files and `BcastInject` source code and recompile java tools (details below).
3. Compile and upload the modified `SimpleCmd` code to the mote.
4. Take mote and sensor board off the programming board and assemble them, then put another mote on the programming board.
5. Compile and upload the `TOSBase` code to the second mote.
6. Run the `SerialForwarder` java application.
7. Send commands to the mote with the sensor board attached using `BcastInject`.

A total of eight code modifications in four files are needed to get the sounder to work. In the `apps/SimpleCmd` directory, we will modify `SimpleCmdMsg.h`, `SimpleCmd.nc`, and `SimpleCmdM.nc`. Then, in the `tools/java/net/tinyos/tools` directory, we will modify `BcastInject.java`.

First, in `SimpleCmdMsg.h`, you add new constants `SOUNDER_ON` and `SOUNDER_OFF` to the enumeration that defines the commands. Second, in `SimpleCmd.nc`, a new wiring is needed:

```
SimpleCmdM.Sounder -> Sounder
```

In order to get the sound function to work, we use the `Sounder` component, which can be found in the `tos/sensorboards/micasb` directory. Note that the name of the last directory will be different depending on what type of sensor board you are using (in our case, `micasb`). Third, remember to include `Sounder` in the list of components at the top of `SimpleCmd.nc`. The fourth change, in `SimpleCmdM.nc`, is to add the line

```
interface StdControl as Sounder
```

to the `uses` clause so that we can control the sounder with the `StdControl` interface. Fifth, in the `switch` statement where the command's action is parsed, we have to add clauses to take care of what occurs when our new commands are used. Our additions are shown below:

```
case SOUNDER_ON:
    call Sounder.start();
    break;
case SOUNDER_OFF:
    call Sounder.stop();
    break;
```

The sixth alteration, the last for the `SimpleCmdM.nc` file, is to add an `init` call to the sounder where the rest of `StdControl` is initiated. To do this we added:

```
call Sounder.init();
```

to the `StdControl.init()` function before the other components are initialized. Finally, we need to make two alterations to the `BcastInject.java` program. The seventh code alteration is to add `SOUNDER_ON` and `SOUNDER_OFF` to the list of byte constants that represent commands at the top of the class. Remember to make sure that the numbers you choose to represent the sounder commands correspond to the numbers you picked earlier in the `SimpleCmdMsg.h` file. The eighth and final code change is in the `if-else` statement that is dealing with the variable `cmd`. Two clauses should be added to deal with our new commands, shown below:

```
} else if (cmd.equals("sounder_on")) {
    packet.set_action(SOUNDER_ON);
} else if (cmd.equals("sounder_off")) {
    packet.set_action(SOUNDER_OFF);
```

With these changes to the code, we've added the sounder's functions to the mote. Using `BcastInject` to send commands to the mote, it should now support all of the commands listed in the previous exercise along with the new commands `sounder_on` and `sounder_off`.

A suggestion for a project is to write a Morse code sounder that would take written sentences and translate them into Morse code, which the sounder would then express. This would involve more work in topics including buffering, custom defined packets, and acknowledgement protocols.

### 3.3 Lab Exercise VI: Data Logging

This exercise is based on lesson 8 in the TinyOS 1.x tutorial, in which we command a mote to record and store data, then transmit it back to the base station PC at a later time.

1. Attach a mote and a sensor board to the programming board.
2. Compile and upload the `SenseLightToLog` application to this mote.

3. Remove the mote and sensor board from the programming board and put them together.
4. Program another mote with the `TOSbase` application.
5. Start `SerialForwarder`.
6. In a new terminal window, use `BcastInject` to command the mote: Type `java net.tinyos.tools.BcastInject start_sensing <num_samples> <interval>` where `<num_samples>` is the number of samples to take, and `<interval>` is the time in milliseconds between samples. We used 16 and 1000, respectively. The mote will toggle the red LED as it takes the readings.
7. After the mote finishes taking the samples, type `java net.tinyos.tools.BcastInject read_log <mote_address>` In our case, the mote address was 1. This command causes the mote to send back the readings that it had stored in memory.

Applications that can store their data for later collection are useful any time sensors need to collect data while they're out of transmission range. One application where this would be particularly useful would be in an agricultural setting, in which sensors could measure the micro-climate, and store the measurements in memory until the data can be collected, for example when a farmer drives within range while working in the field.

## 4 Discussion

While we were unsuccessful installing TinyOS 2.x manually on Linux and Windows/Cygwin, we recently had success running it from a live CD, which is a pre-made bootable Linux CD. TinyOS 2.0.1 was released in May 2007, and the tutorials and installation instructions are still somewhat incomplete. We are currently working on getting TinyOS 2 to work. When we accomplish this, our next step will be to work on some of the new tutorials that come with TinyOS 2, such as an anti-theft application.

## 5 Conclusion

This paper extended our earlier paper and described three additional exercises about remotely controlling motes from a PC. Combined, our exercises cover all of the basic TinyOS abstractions: computation, communication, sensing, and storage.

With such a broad range of applications, sensor networks are an important emerging technology. Because of this great potential, it is very likely that sensor networks will become much more commonplace in the near future. This will create a demand for people who have the skills for, and familiarity with setting up and working with sensor networks.

The exercises described in this paper and the previous paper attempt to reconcile sensor networks and undergraduate education. Hopefully, these exercises will help sensor networks to become an engaging and frequently taught topic in the undergraduate classroom.

## Acknowledgements

This work is supported in part by the W. M. Keck Foundation, the John S. Rogers Program, and by the National Science Foundation under grant CNS 0720914. We would like to thank Chris Allick, John Charnas, Alex Hickman, Damon Tyman, Jeffrey Springer, and William Sehorn.

## References

- [1] MIT Technology Review. *10 Emerging Technologies That Will Change the World*. Feb 2003. <http://www.technologyreview.com/Infotech/13060/page2/>
- [2] *TinyOS tutorials*. <http://www.tinyos.net/tinyos-1.x/doc/tutorial/>
- [3] *TinyOS Mission Statement*. <http://tinyos.net/special/mission>
- [4] Wired Magazine. *Intel's Tiny Hope for the Future*. [http://www.wired.com/wired/archive/11.12/intel.html?tw=wn\\_tophead\\_5](http://www.wired.com/wired/archive/11.12/intel.html?tw=wn_tophead_5)
- [5] Crossbow, Inc. <http://www.xbow.com/>
- [6] *Introduction to NesC*. <http://nesc.sourceforge.net/>
- [7] Jens Mache, Chris Allick, John Charnas, Alex Hickman, and Damon Tyman. *Sensor Network Lab Exercises Using TinyOS and MicaZ Motes*. Proceedings of the International Conference on Pervasive Systems & Computing, 2006. <http://www.lclark.edu/~jmache/jm/worldcomp06.pdf>