

CS344E Assignment 1: Discovery

Due: 11:59 PM, April 16, 2006

Basics

In this assignment, you'll get up to speed using TinyOS and Telos motes by writing a simple low-power neighbor discovery protocol. Your protocol has a single parameter, the duty cycle of a node (the percent of the time it spends awake). You'll write an application that uses your protocol to have nodes blink their LEDs in a synchronized fashion. You may do the assignment either in groups of two or individually.

Installing TinyOS

To complete this assignment, you'll need to install TinyOS on a machine that has USB ports, so you can program your Telos nodes. You'll be using TinyOS 2.0. To download TinyOS and all of its tools (the nesC compiler, msp430-gcc, etc.), go to <http://www.tinyos.net/tinyos-2.x/doc/html/install-tinyos.html> and follow the instructions there.

Getting Started

TinyOS 2.0 has four forms of documentation:

1. **Tutorials** are simple introductions to TinyOS 2.0 and how to program it. Right now they're a bit sparse. The only two that are fully fleshed out are Lesson 1, which describes the component model, and Lesson 5, which describes communication. The tutorials can be found in the `tinyos-2.x/doc/html/tutorial` directory.
2. The **Programming Manual** is a more in-depth discussion of how to program in nesC. It talks about the basic programming constructs and how to use them. You can download the manual: <http://csl.stanford.edu/pal/pubs/tinyos-programming-1-0.pdf>.
3. **TEPs** provide detailed descriptions of the various parts of a TinyOS system, including the boot sequence, communication, power management, timers, and sensing. While the programming manual tells you how to use nesC to write effective code, TEPs are more akin to a description of the TinyOS APIs. HTML of the TEPs can be found in the `tinyos-2.x/doc/html/` directory.
4. **nesdoc** is the nesC equivalent of javadoc: it is direct documentation of the TinyOS source code. The best way to sift through nesdoc is to start at `tinyos-2.x/doc/html/nesdoc/telosb`.

You might find the TOSSIM simulator useful. It currently only supports the micaZ platform, but as long as you are not using MSP430-specific interfaces (which have Msp430 in their name), this should not be an issue. The TOSSIM tutorial can be found in `tinyos-2.x/doc/html/tutorial/lesson-t.html`.

The `tinyos-help` mailing list archives are an excellent resource for answering questions. If you can't find your answer there, don't be afraid to ask.

Assignment

Your protocol takes a single parameter, the node duty cycle. An application must be able to specify this parameter at the command line by defining `DISCOVER_DUTY` at the command line. The parameter is an integer in the range of 1-1,000 whose inverse is the node duty cycle. E.g., `-DDISCOVER_DUTY=100` will set a duty cycle of $\frac{1}{100}$, or 1%. You should turn the radio on and off with the `SplitControl` interface of `ActiveMessageC`.

You cannot assume any initial synchronization between the nodes, and nodes may join or leave freely. You can assume that neighborhoods are fully connected, and no node is a member of more than one neighborhood. Given these constraints, you must write a protocol that will let a node discover all of the other nodes in its neighborhood. Nodes within a neighborhood should turn on their LEDs in a synchronized fashion, where in an n node neighborhood each node has its LED on $\frac{1}{n}$ of the time and no two nodes have their LEDs on at the same time. The periods over which nodes have their LEDs on do not have to be regular: all that matters is *fairness*, so each node is on $\frac{1}{n}$ of the time. *Latency* is also important: once a neighborhood is discovered, no node should go for `DISCOVER_DUTY` seconds without having its LED turn on. Your discovery protocol only needs to scale to neighborhoods of 8 nodes.

You should not make any modifications to any TinyOS core components (components in `system/`, `lib/`, `chips`, or `platforms`). All of your code should live in a single application directory.

Handing In

All of your code should be in a single application directory. It should be reasonably documented (e.g., explain, at a high level, what each function does if it's not very simple). The directory must have a `README` that describes your algorithm and how it works. If you have any TOSSIM test scripts, include those as well, with a description of their expected behavior in comments. Send a tarball of this directory to Jung Woo.