CSE 160 – Computer Networks

University of California, Merced

Project 1: Flooding and Neighbor Discovery

Introduction

In this assignment, you are to develop an application in TinyOS to send message between nodes through flooding. Also you will implement neighbor discovery

Objectives and Goals

There are two main objectives to this project:

Flooding

Each node floods a packet to all it neighbor nodes. These packets continue to flood until it reaches its final destination. Must work as pings and ping replies. Only use information accessible from the packet and its headers.

Neighbor Discovery

Each node should be able to discover all of its neighbors. Accomplish this without using a new packet type than what is provided. Account for neighbors that drop out of the network.

Where to Start

As you have learned from the previous section, TinyOS is compiled from .nc files. To start this project, we provided you with some skeleton code.

In the provided skeleton code, a node is only able to send a ping packet to a neighbor node. If you tell a node to send a packet to a node that is not directly connected to the source, the sending will fail. Your job is to ensure that the packet will reach their destination by storing the destination (and other important information) in the packet header and using a simple flooding system. This header is included in the skeleton code.

Requirements

To illustrate properly implemented flooding, make sure to use the debug channel FLOODING_CHANNEL to showing how your implementations works. (see

includes/channels.h). This channel should print a small message whenever a packet is received or sent, stating the location it was sent from.
Similarly, for the second portion of the project use the channel NEIGHBOR_CHANNEL. This channel should be capable of telling its neighbors when a command packet is issued.

Deliverables

What is expected on the due date is the following:

- Source code of the TinyOS implementation with working flooding and neighbor discovery
- A single page document describing the design process and your design decisions.
- A document with short answers to the discussion questions.

A physical copy of the document and related questions is required on the due date. All documents and a copy of the source code should be submitted to the class web page. Please create a compressed tarball such as Student-Name-proj1.tar.gz. You must do this before class on the day that it is due.
Additionally, you will need to demonstrate that the code you submitted works in the next lab, and be able to describe how it works. Also discuss the design process in the code.

Troubleshooting, Common Problems, and Hints

Hints for Flooding

- Packets should not circulate indefinitely. Your nodes should keep track of what packets it has received. The sequence number and source address in the header file are a good place to start for identifying variables. Implementations that have packets that circulate indefinitely will be considered incorrect.
- Sequence numbers should be increased when sending a new packet out.
- No prior knowledge of the topology should exist when using flooding. A network map will be implementing in the next project.
- To broadcast to all nodes use the constant broadcasting address (AM_BROADCAST_ADDR).

Hint for Neighbor Discovery

- Do not use a separate packet type. This entire task can be done using a combination of ping, ping reply, flooding, and TTL.
- Neighbor discovery can be implemented using a timer, and run continuously in the background.
- Be careful not to overload the network and find a sufficient period for your timer.

Discussion Questions

1. Describe a pro and a con of using event driven programing.
2. Flooding includes a mechanism to prevent packets from circulating indefinitely, and the TTL field provides another mechanism. What is the benefit of having both? What would happen if we only had flooding checks? What would happen if we had only TTL checks?
3. When using the flooding protocol, what would be the total number of packets sent/received by all the nodes in the best case situation? Worse case situation? Explain the topology and the reasoning behind each case.
4. Using the information gathered from neighbor discovery, what would be a better way of accomplishing multi-hop communication?
5. Describe a design decision you could have made differently given that you can change the provided skeleton code and the pros and cons compared to the decision you made?

Grading Guidelines

Each part of the project is graded on a 5 point (0-4) scale, multiplied by the weight of the project. The weighted grades from all parts of the project are added together to produce the final grade.

The five point scale is based on how well you show your understanding of the problem, and in case of code how well your implementation works:

0 – nothing turned in

1 – show minimal understanding of the problem / most things don't work

2 – show some understanding of the problem / some things work

3 – show a pretty good understanding of the problem / most things work

4 – show excellent understand of the problem / everything works

The weights for Project 1 are:

40% - Flooding implementation

50% - Neighbor discovery implementation

5%   - Write-up design decisions

5%   - Discussion questions

Your submission will be graded on correctness, completeness, and your ability to explain its implementation.

<u>Information about the Skeleton Code and TinyOS</u>

Receive and Rebroadcast

With the initial skeleton code provided, nodes are able to communicate to directly connected neighbor nodes, but not over multiple hops. This is problematic since nodes that are not directly connected won't be able to send messages to each other. The goal, however is to be able to do multiple hops.

When a message is received the receive event is signaled. Within this event you can place additional code in order to allow for multi-hop.

Sending Queue

The radio is capable of only sending one packet at a time. If there is already a packet being sent, the sending of a packet will fail with an EBUSY error returned. In order to keep these packets from simply being dropped, a simple lock is introduced and a first-in-first-out queue. This system works by requesting to post a task. A task will be added to the event handler and executed in a round robin fashion.

Timers

Timers are a good way of running a piece of code periodically. To implement your goals in your project, it may be helpful to understand how to wire and use a timer in TinyOS. To add a timer you must declare a new component and wire it to an interface in your main function.

```
// NodeC.nc implementation {
components MainC;
components TimerMilliC() as myTimerC; //create a new timer with alias “myTimerC”

App.Boot → MainC.Boot;
App.periodicTimer → myTimerC; //Wire the interface to the component
}
```

```
//Node.nc
uses {
interface Boot;
interface Timer<TMilli> as periodicTimer; //Interface that was wired above.
}
```

To use the timer you can simply call a command. “call periodicTimer.startPeriodic(100)” will start the timer to run every 100 milliseconds. When the timer fires the event “fired” will be signaled. Events must be implemented and given something to do.

```
//Node.nc
event void periodicTimer.fired(){
//Do necessary task
}
```

It is important not to run all the timers at the same time or else they will interrupt each other or cause other types of congestion in your network. To avoid this it is best to use a random number and a prime number. See the Random interface.

Variables in nesC

Allocation in nesC works different than it does in normal C and C++. All variables should be statically allocated and not manually allocated by functions such as malloc. In most situations it is best to get a pointer from the variable to further manipulate it.

```
//Example uint16_t
data;
manipulation(&data);
```

A rule of TinyOS is that all variables should be declared at the start of a function, if, while, case, or for statement.

```
// Correct Example
if ( flag == TRUE){
uint16_t var1;
uint16_t var2;

var1 = 1;
var2 = 2;
}

// Incorrect Example
if ( flag == TRUE){
uint16_t var1;
var1 = 1;            // This line will cause an error

uint16_t var2;
var2 = 2;
}
```