Fishnet Assignment 1: Random Forwarding

Due: Thursday, Jan 24, 2002 at the beginning of class. Out: Monday, Jan 7, 2002.

CSE/EE461 Winter 2002; Anderson.

In this assignment, you will work in teams of two to develop a single C program that is a Fishnet node. The node will connect to other Fishnet nodes and allow you to send simple messages that you type on the keyboard, such as "Hello World", to other nodes in the network. The goals of this first assignment are to become familiar with the Fishnet development environment and to understand packet forwarding.

1 Introduction to the Fishnet

Fishnet is a class-sized network that we will build over the course of the quarter! You will build a Fishnet node progressively, working in pairs, over a series of four assignments. Once we are done, nodes from each team will be able to route messages and transfer files in a shared, secured network.

1.1 Development Environment

We will be developing under Linux and writing C code. Typically, you will go the CSE Labs, log into a Windows machine, and from there remotely access a Linux server using teraterm (ssh) or X windows. All programming, compiling and running of the Fishnet will take place on the Linux machines. Each node of the Fishnet network will run as a process on a Linux machine. The compiler we will use is gcc, and the debugger we will use is gdb. You can use any editor you prefer, such as emacs.

1.2 What is a Fishnet?

The term "Fishnet" is somewhat overloaded, and before going further we want to clarify what it means. First, it is the code that makes up all of the assignments. This is the *Fishnet development environment*. Second, it is a running network of many nodes and a single fishhead (a key component to be described shortly). This is what we most commonly mean by a *Fishnet*. Note, however, that there are many potential Fishnets. You will all start your own, private, Fishnet to develop and test the code required for this assignment. Each of you will run your own fishhead, and nodes of your network will not interact with nodes of other students' networks. There is also The Fishnet, with a capital "T". This is the one public, shared Fishnet (a running network) that everyone's node is able to join and participate in a collective class network. This is accomplished simply by pointing your code at a fishhead that we run.

1.3 Fishnet Components

The Fishnet project code, like everything else you need, is available from the course web site. The package we provide you with contains the following key components:

1.3.1 fishhead

fishhead is a program that manages Fishnet nodes. A network can contain many nodes, but only one fishhead. The main function of the fishhead is to tell individual nodes who their neighbors are. It is important to understand that the fishhead manages the network topology and decides who is connected to whom, not you in your programs. (For the curious, the nodes of networks you operate are run as separate processes that communicate with each other using a UDP overlay.)

When you develop a Fishnet node and run it, one of the first things it will do is to join a Fishnet network by contacting the fishhead. This means that before you start one of your Fishnet nodes there must be a fishhead process running. You only need to start a fishhead for your network once, even though the nodes in the network can come and go.

1.3.2 libfish.a

The fish library implements all of the Fishnet functionality that you will need for your assignments. When you send a message using the fish_sendframe() function, for example, libfish.a is called to do the work of sending the packet. libfish.a also prints a large (but controllable) amount of debugging information to the console (stderr) to help you understand what is going on with your program. The library source code is available in the fishsrc directory so that you can see how the Fishnet really works and to help with your debugging, but you MUST NOT change this code at all so that you remain compatible with other people's nodes.

1.3.3 fish.h

This is the header file that you should include in your C program to gain access to the functionality implemented in the fish library. It contains the dozen or so Fishnet API functions that you can call, as well as the structures that define packet formats, and other Fishnet constants. You should read the comments in this file, as it contains the definitive Fishnet API documentation.

2 What You Need To Do

Write a C program in a file called hw1.c that, when run, does the following:

- Takes three whitespace separated command line arguments: first,a string representing the location of the fishhead, in the format "hostname:port"; second, the name of the domain which your node will belong to, which is simply a string and can be anything for this assignment; third, the address you want for your Fishnet node, which is simply a small unique number.
- Joins the Fishnet described by the command line arguments. You will need to use the fish_joinnetwork() call, and you will need to have started your own fishhead before running your program. Remember, you choose the address of your node, but the fishhead decides what other nodes you are connected to, and your neighbors change as other nodes come and go.

- Waits to get a line of input from the keyboard using fish_recvhook() and checks to see if the input is of the form "send <address> <message>", where <address> is a number representing the address of the node to send to, and <message> is a string that may contain spaces and is terminated by the end of the line. You can use the sscanf code "send %d %a[^\n]" to parse the input. The "%a[^\n]" accepts all characters up to the end of the line, unlike "%s", which stops at the first space. Free the string in which you store the message when you are done with it. You can use the command "`man 3 sscanf`" to learn more about this.
- When a command of this form has been received, your program should send a packet containing the message to a random neighbor using fish_send(). This requires that you construct a packet structure (`struct packet`) and fill in the source address, destination address, TTL, protocol, and packet contents as appropriate. The value of the TTL should be the constant MAX_TTL, and the protocol should be FISH_PROTOCOL_ECHO_REQUEST. These constants are defined in fish.h. The rest of the program described below will forward the packet through the network to its destination and cause an acknowledgement packet to be returned to the sending node.
- Waits to receive a packet from the network using fish_recvhook(). When a packet is received, you should deal with it as follows:
    - If the packet's destination is not your node, you should decrement the TTL field. If the new value of the TTL is greater than 0, you should forward the packet to a random neighbor. Otherwise, the packet's time to live has expired; don't do anything more with it.
    - If the packet's destination is your node and the protocol is FISH_PROTOCOL_ECHO_REQUEST, you should print "Echo request from <address>: <message>\n", where <address> is the source of the packet and <message> is its data. Then you should make a new packet with the protocol FISH_PROTOCOL_ECHO_RESPONSE and send it back to the node that sent you the original packet. The packet should have the original message as its data.
    - If the packet's destination is your node and the protocol is FISH_PROTOCOL_ECHO_RESPONSE, you should print "Echo response from <address>: <message>\n".
- Your program should repeat getting input, sending, receiving, and printing indefinitely. (When you type the command "exit", libfish.a will automatically end the program.) Your program must be capable of sending and receiving in any order; this will happen without any effort on your part if you are using upcalls in the right way.
- An important rule of thumb in building network protocols is the **Robustness Principle**: "Be conservative in what you send, and liberal in what you receive." This helps different implementations of a protocol (e.g., the sample solution, your program, and your classmates' programs) to work together reliably. For this assignment, this principle means you should be careful to send packets that comply with the protocol we have described, but you should do the best you can with

whatever packets you receive from other nodes. In particular, other nodes shouldn't be able to crash your node by sending it bad packets. If your node does crash, it's your responsibility to find out what happened and fix it.

3 Step By Step Instructions

You may develop your program in any fashion you prefer. Here is a suggested set of steps to develop the required functionality.

0. Check that you can get into the CSE Lab and log into a Windows machine. From there, check that you can remotely access one of the Linux servers (`fiji`, `tahiti`, `sumatra`, `ceylon`, `blackbox02`, `blackbox03`, `blackbox04`, or `blackbox05`). If you are not a CSE major, you must fill out a request form in order to gain access and accounts; the form will be available in class or at the CSE front desk. All of your work will take place on the Linux servers. The easiest way to work is to use teraterm to ssh to these machines; there is a shortcut to do this on most desktops. You can also use any X windows packages that you are familiar with.

1. Download the Fishnet package from the course web pages into your home directory and unpack it. The easiest way to do this is to run the lynx browser (`lynx http://www.washington.edu/461`), follow the Fishnet links, and download the tarball. Alternatively you could use `wget` to download the tarball. To unpack the tarball, use `tar` with the arguments given on the web page. You should now have a fishnet directory with files in it. As a general note, you can use the `man` command to get a help page for any Unix command or function we ask you to use that you don't understand.

2. Read `fish.h`. The comments in this file tell you what the Fishnet API calls are, what they do, what arguments they take and return, and so forth. You won't find this information anywhere else.

3. Start a fishhead process to manage a Fishnet by running the fishhead program that is inside the fishnet directory. Type "`./fishhead --help`" to find out what command line arguments it accepts. You will need to give it a argument to indicate which port it should listen on for messages. Pick a number between 1024 and 32K. If the number you choose is the same as someone else's fishhead will fail to start. The fishhead program runs indefinitely to keep the Fishnet it is running up. You should leave it running and create another window in which to do your development. When you're done, stop it by typing ^C.

4. Download the sample executable for this assignment from the course web site and try running it with your local fishhead. You'll need to run it more than once, in separate windows, to do anything interesting.

5. Develop your program, which should be contained in a file called hw1.c in the top level directory created when you unpacked the Fishnet distribution. The fishsrc directory contains the source code that we have written so that you may see it and use it for debugging, but you must not change it. We have supplied a Makefile so

that you can type "`make hw1`" to compile your program. This will invoke the compiler, gcc, with the right command line arguments.

- You will find the following C library functions helpful: atoi, printf, sscanf, strlen, strncpy, and free. Remember than you can type "`man <functionname>`" to get help for using these functions.
- As you write your program, note that our (fairly verbose) sample solution is about 130 lines. If your program is much longer than this, you are probably doing something the hard way and should talk to us about it.

6. Test your program by running it and the sample solution to make a network with a few nodes that are connected to one another. You will need to use different command line arguments to select different node addresses. You will find it easiest to bring up a separate window for each node, as well as the window for the fishhead. If your program works, you should be able to enter text in one window, see it echoed in another window, and see the response back in the first window. Unless you've disabled them using the fish_setdebuglevel() function, you should also see libfish debugging messages that show how your packets move through the network.
7. Iterate on the above two steps until you think you have your solution! Remember that only feasible way to develop software of any complexity is to find a way to break the task down into simpler, checkable, subtasks. In this case, you might first write a program that had a main() function that simply accepts the command line arguments, prints them out, and exits. When this compiles and runs properly (experience shows most of you will have found and fixed at least one error already!) move to the next task. The next task might be joining the network and sitting in fish_main(). You can use debugging messages to check if this program has worked, as you will get messages when you have joined. You can quit the program by typing the command "exit". Next, you might try for keyboard input, and print it out but not send a packet yet, and so forth. While this strategy might seem like overkill at this stage, you should get used to it before attempting the larger assignments. The beauty of testing subtasks is that the problems are rarely where you think they are (otherwise you wouldn't have made the mistake) and testing is the only way to pin them down. Despite this strategy, more difficult bugs will require you to use the debugger, gdb.
8. Comment your program if you haven't already. Good comments don't belabor the obvious (e.g., "calling the main loop" near fish_main()). Rather, good comments tell us how you have arranged your code and assumptions you have make, as well as anything non-obvious (e.g., "this is the frame receive upcall from fish_main" near the function you installed using fish_recvhook()). In truth, this first assignment shouldn't need many comments, but in later assignments we will need comments from you to understand your code.
9. Join the class Fishnet! The fishhead you should connect to is at jimbo:7777. You can see the network topology by opening http://jimbo.cs.washington.edu:7777/ in your web browser. Try sending some messages to the other nodes you find there and make sure you can deal with the responses you get.

10. Answer the discussion questions and follow the turn-in instructions below.

4 Discussion questions

To prepare for turnin, you need to gather the output of your program running the test case below and answer two discussion questions.

1. Run a three node network (using your program only, not the sample solution), with each node running in a separate window. From one node, send a message to each of the two other nodes. Capture the entire output of the three sessions using, for example, the `script` command. Make sure the debugging level is FISHNET_DEBUG_ALL (the default) so that we can see what packets are being sent and received. Mark up the printout to tell us how the output lines correspond to the send commands.
2. Why do we need the TTL mechanism, given that random forwarding ensures that a packet sent by one node will eventually reach any other connected node?
3. Packets can bounce around for some time with random forwarding, particularly in large networks. Your friend suggests a modification: never send a packet back to the neighbor it just came from. For what kinds of network topologies would this be helpful, and for what kinds would it be harmful?

5 What To Turn In

For this assignment, as well as for future assignments, you need to turn in electronic and paper material as follows.

1. Join the class Fishnet using the command "`./hw1 jimbo:7777 <domain> <address>`". Then enter ^D (Ctrl-D) to close standard input, ^Z to suspend the program, and finally the command "bg" to turn it into a background process. This will allow your program to keep running even after you log out, so that your node can interact with other nodes in the network. Your program should still be running and forwarding packets when we grade the assignment. You may want to check on it from time to time by using the `ps` command or by looking at the Fishnet topology web page at http://jimbo.cs.washington.edu:7777/. If your program crashes, you will find a core file in your development directory that you can examine using gdb. You can use the `kill` command if you need to restart your node. We will tell you when to kill your program after we are done grading the assignment.
2. Join the class Fishnet again (using a different address) and send messages to node 1 saying that you and your partner were there. Node 1 will maintain a log of these messages. You can see the message log by looking at the web page http://jimbo.cs.washington.edu:8000/.
3. Use the `turnin` program on the Linux servers to electronically submit one or more C files containing the source code of your solution. In this case, the main file should be hw1.c. You must do this before class on the day that it is due. The

code you send should be suitable for us to manipulate automatically for grading checks. (In particular, it should compile using the makefile we provided.)

4. In class on the due date, hand in one stapled paper write up, with both partners' names on it, containing:
    a. A printout of the source code you submitted electronically.
    b. A printout of any output we have asked you to capture. In this case there are three output files captured with `script`, one for each of the three nodes running the test case.
    c. Short answers to the discussion questions.

6 Our Grading Philosophy

Our intent is to have you explore networking issues, rather than write a program that manages to pass a given set of acceptance tests. We will grade you on this material in two ways. First, we will look at your write up and the C source code. The printout, your description of what is going on, your answers to the discussion questions, and the design clarity of your program, give us a very good idea of how well you got the program working and how well you understand the issues that it explores. Second, we may look at how your program is behaving in the class Fishnet. Third, we may compile and run programs that we are unsure of after looking at the source and your printout. We also may randomly test programs to see that they produce the output that you claim they produce. When we test, it isn't our intent to penalize you significantly for small errors in execution, as opposed to larger errors in understanding of the concepts and design of the solution, but we will deduct some points for small errors or not following our instructions. We will also sometimes include optional material in the assignments, and hope that you choose to explore some of the issues this material raises, but we won't grade optional material.