# Fishnet Assignment 2: Random Forwarding

**Due: Monday Oct. 22, 2001 at the beginning of class. Out: Monday Oct 8, 2001.**
**CSE/EE461 Autumn 2001; Wetherall.**

In this assignment, you will work in teams of two to develop a single C program that is a Fishnet node that forwards echo request packets to a destination across the Fishnet and forwards subsequent echo reply packets back across the Fishnet. The program you write builds on your solution to the first Fishnet assignment. The goal of this assignment is for you to understand packet forwarding.

## 1 The Program You Need To Write

Write a C program in a single file called hw2.c that, when run, does the following:

- Takes three whitespace separated command line arguments and joins the Fishnet described by those arguments. This is identical to the first assignment.

- Waits to get a line of input from the keyboard and checks to see that the input is of the form "`send <nnn> <message>`", where `<nnn>` is a number representing a node address and `<message>` is a message to send. You can use the sscanf C library function parse an input string into these components. Then, your program should send a packet containing the message to a randomly selected neighbor node. This requires that you construct a packet structure (`struct packet`) and fill in the header fields such as source address, destination address, TTL, and protocol and the contents as appropriate. The value of the TTL should be the constant MAX_TTL, and the value of the protocol field should be the constant FISH_PROTOCOL_ECHO. These constants are defined in fish.h. The function `random`() generates a random number, which you can use as part of selecting a random neighbor. When you send the message, print "`Send (src=<S>, dst=<D>, ttl=<T>, prot=<P>) <message>\n`" where the parameters in angle brackets are the values in the corresponding packet fields and '\n" means a newline. 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.

- Wait for a packet to be received from the network. When it is, you should check the destination address to see if it is intended for your node or some other node and handle it accordingly:

    - If the packet is not intended for your node, you should decrement the TTL field. If the new TTL value is zero you should not forward the packet but just print "`Zero (src=<S>, dst=<D>, ttl=<T>, prot=<P>)\n`", again where the parameters in angle brackets are the values in the corresponding packet fields and '\n" means a newline. If the new TTL is not zero you should send the packet to a randomly selected

neighbor and then print "`Forw (src=<S>, dst=<D>, ttl=<T>, prot=<P>)\n`".

- If the packet is intended for your node and the value of the protocol field is FISH_PROTOCOL_ECHO, you should print "`Recv (src=<S>, dst=<D>, ttl=<T>, prot=<P>) <message>\n`". Then you should then make a new echo response packet and send it back to the node that sent you the original packet. The value of the protocol field should be FISH_PROTOCOL_ECHO_RESPONSE and the contents should be empty. After sending the response packet, print out "`Send (src=<S>, dst=<D>, ttl=<T>, prot=<P>) ECHORESPONSE\n`".

- If the packet is intended for your node and the value of the protocol field is FISH_PROTOCOL_ECHO_RESPONSE, you should print "`Recv (src=<S>, dst=<D>, ttl=<T>, prot=<P>) ECHORESPONSE\n`".

- You must write only one program in the single file hw2.c, rather than separate programs for sending, receiving, and forwarding, and your program must be capable of getting input and sending, forwarding, and receiving and echoing in any order. The latter will happen without any effort on your part if you are using upcalls in the right way. Your program should repeat getting input, sending, forwarding, and receiving indefinitely. When you enter ^D (end of file) libfish.a will automatically end the program.

# 2 Step-by-Step Development and Test Instructions

You can go about building your program in any fashion you prefer. Here is a suggested set of steps to develop the required functionality. We have omitted the steps up through and including starting the fishhead, which remain as in assignment 1.

0. Start with hw1.c by copying it to the new file hw2.c

1. Write the code using sscanf() to parse the keyboard input into a "`send <nnn> <message>`" command, and just send the message directly as before. If you run this with a two node network you should see the message being printed out when it is received.

2. Write the code to fill in and send a packet structure with the message as the contents. On the receive end, you will have to interpret the packet structure to see what you received. You should now be able to change the print messages from assignment one to those specified above for assignment two. Again, test this with a two node network.

3. Write the code to send a packet to a random neighbor (rather than all neighbors or the first neighbor). You can generate a random number between 0 and N-1 with

the expression "`random() % N`". Now test that your program works with a three node network.

4. Write the code to check the packet header to see if a received packet has arrived at its destination and if not to forward it to a random neighbor, printing the required message. The distinction between echo request and response packets is not relevant for forwarding. Again, test that a packet bounces around the three node network until it reaches its destination and then stops.

5. Write the code to decrement the TTL value during forwarding and discard packets when their TTL reaches zero, including printing out the required message. Test this code by running a three node network and temporarily changing your program so that the initial TTL value is one.

6. Write the code to generate and send an echo response packet when an echo request packet arrives, including the print messages for sending and receiving echo responses. Test your program on a three node (or larger) network. Comment your program if you haven't already. Your program should now do everything it needs to do!

7. *For Fun*. Test your program in a three node (or larger) network that includes nodes running our reference solution, which we will release shortly. This test is just to make sure that your nodes are interoperable with our nodes. Now change your command line arguments to run one of your nodes that joins the class Fishnet. To see what other nodes are part of the class Fishnet at the moment, use your browser to download a page from the fishhead running on port 7777 of galah.cs.washington.edu, e.g,. http://galah.cs.washington.edu:7777/. You should be able to join, send packets to other nodes, receive replies from them, and forward packets on behalf of other nodes!

# 3 Turn In and Discussion Questions

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

1. Run a three node network, with each node running in a separate window. For each node in turn, send one packet to each of the other two nodes (for a total of six send commands). Capture the entire output of the three sessions using, for example, `script`. Mark up the printout to tell us which output lines correspond to which of the six send commands.

2. Question: Why do we need the TTL mechanism, given that random forwarding ensures that a packet sent by one node eventually reaches all other connected nodes?

3. Question: Packets can bounce around for some time with random forwarding, particularly in large networks. You friend suggests a modification: don't forward

a packet back to the neighbor it just came from. For what kind of network topologies would this be helpful and for what kind of network topologies would this be harmful?

You should turn in one copy per team of electronic and paper material as usual:

1. One C file called hw2.c containing the source code of your solution. Submit this electronically using the `turnin` program on the Linux servers.

2. One stapled paper writeup with your names that contains:
    a. A printout of the annotated output from the test case described above.
    b. A printout of the source code you submitted electronically.
    c. Short answers to the discussion questions above.

—END—