Fishnet Assignment 3: Distance Vector Routing

Due: Monday Nov. 5, 2001 at the beginning of class. Out: Monday Oct 22, 2001.

CSE/EE461 Autumn 2001; Wetherall.

In this assignment, you will work in teams of two to develop a Fishnet node that forwards echo request and reply packets across the Fishnet using routing tables that you maintain. The program you write builds on your solution to the second Fishnet assignment by forwarding packets according to a routing table instead of randomly. The goal of this assignment is for you to understand distance vector routing.

1. What You Need To Write

Write a C program in a single file called hw3.c that does the following:

- Takes three command line arguments and joins the Fishnet described by those arguments. This is identical to previous assignments.

- Waits to get a line of input from the keyboard of the form "`send <nnn> <message>`", then sends an echo request packet across the Fishnet to the destination, from where an echo reply packet is sent back to and received at the original sender. This is identical to the second assignment, and the output of your program for packet sending, forwarding and receiving should also be identical in form to the second assignment.

- Maintains a routing table which is an array of MAXADVERTISEMENTS routing table entries. You must define the structure of a routing table entry. It should have several types of information: a destination address, the best distance to that address, the preferred neighbor, and the age of the information.

- Maintains a periodic timer that fires every ten seconds. When the timer fires you should print "Periodic Update\n", age the information in the routing table, and then send routing updates to all neighbors. Both of these steps are described below:

    - To age the routing table, you should look for entries that have not been refreshed (as described further down) for the past three consecutive timer intervals. These entries have expired and must be removed from the routing table. Each time you remove an entry you should print "`Route expire <D, cost M, via N>\n`" where D stands for the destination address removed, M its distance metric, and N the preferred neighbor.

    - A routing update is a packet with a protocol value of FISH_PROTOCOL_ROUTING whose contents is an array of routing advertisements as defined in fish.h. It should include one advertisement for the address of the local node at a distance metric of zero, plus one advertisement with address and distance for each entry in the routing table. (These entries have been learned from neighboring nodes as described below.) After sending each routing

update you should print "`Route send to N, X adverts\n`", where N is the neighbor being sent the update and X is the number of advertisements that the update contains.

- Receives routing update messages from neighboring nodes and uses them to update and refresh the contents of the routing table as follows. Each time you receive a routing update you should print "`Route recv from N, X adverts\n`", where N is the neighbor that sent the update and X is the number of advertisements that the update contains. Routing advertisements carried in the routing message are processed in turn as described below, where each advertisement is for a destination D at distance metric M learned from preferred neighbor N. Each entry for which a message is printed is refreshed for the purpose of aging.

    - If D is not in the routing table and is not the node address, then it is added at distance M+1 via neighbor N and the age is set to zero. In this case print "`Route add <D, cost M+1, via N>\n`".

    - If D is in the table with neighbor N and distance M+1 then print "`Route refresh <D, cost M+1, via N>\n`".

    - If D is in the table with neighbor N or distance greater than M+1 then the entry is updated to be at distance M+1 via neighbor N. If the distance is now INFINITY or greater (defined in fish.h) then the routing entry should be removed and you should print "`Route infinity <D, cost M+1, via N>\n`". Otherwise you should print "`Route change <D, cost M+1, via N>\n`".

- Forwards packets received from neighboring nodes that are destined for other nodes, decrementing the TTL as before. Instead of sending the packet to a random neighbor, you should send it to the preferred neighbor listed for the destination in the routing table and print "`Forw (src=<S>, dst=<D>, ttl=<T>, prot=<P>)\n`" as before. If the destination is not in the routing table then you should print "`Unreachable (src=<S>, dst=<D>, ttl=<T>, prot=<P>)\n`" and drop the packet.

- As before, your program must perform all tasks in any order and run until you enter ^D (end of file), when the libfish.a will end the program.

1. 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 previously.

0. Start with hw2.c by copying it to the new file hw3.c

1. Leave the program performing random forwarding and add the code for the periodic routing timer. Look at fish.h to see the timer API calls. When the timer fires, send a routing update message with one advertisement (for the node itself) to all neighbors and print the required messages. Test this by running both a two and three node network and

seeing that routing updates are exchanged.

2. Define the structure for a routing table entry and add the routing table. Change the random forwarding routine to forward using the routing table and print out the forwarding related messages. Test this so far by running a two node network. You should not be able to send any non-routing messages (they should all be dropped because there is nothing in the routing table).

3. Write the code to add new advertisements in routing updates to the routing table and to send routing updates that encode the information in the table. Test this with a two and three node network. You should see routes added at each node for the other nodes, and forwarding should have begun to work.

4. Write the code to handle the remaining routing update cases. Test your program on a two node network. Routes should be added, and refreshed too.

5. Write the code to age entries in the routing table and expire them when they get old. Test this with a two node network by letting routing stabilize and killing one node. The other node should eventually expire its route to the killed node.

6. At this stage you have a complete program and should do the turnin cases.

7. *For Fun*. Test your program for interoperability with the class reference solution and then join the class Fishnet (via the fishhead at port 7777 of galah.cs.washington.edu) to bounce packets off other team nodes.

8. *For Fun.* (Do this after you have saved the turnin output.) Speed up route convergence by adding triggered updates. Whenever a received update changes your routing table, then immediately send updates to your neighbors. Try this on the turnin test case to see the difference.

9. *For Fun*. (Do this after you have saved the turnin output.) Speed up route convergence by adding the split horizon with poison reverse heuristic. When sending an advertisement to a neighbor, note which advertisements were learned from that neighbor in the first place. Send these routes with a metric of INFINITY. Try this on the turnin test case to see the difference.

1. Turn In and Discussion Questions

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

1. Run a three node network, with each node A, B, and C running in a separate window. Wait until the routing tables have stabilized and send one packet from A to B. Then kill node B (with a ^C) and wait for the routing tables to stabilize. Try sending from A to B and observe the result. Now restart node B, wait for the routing to stabilize, and try sending again. Capture the entire output of the three sessions using, for example, `script`. To deal with the typing input while your program is producing output, you may want to cut and paste the "send" command.

2. Question: What happens after the node B fails and why? Describe the general progression of your output for nodes A and C in no more than three sentences.

3. Run a three node chain network by starting the fishhead with the "--topology"argument and then starting nodes A, B, and C in that order. Node B should be in the middle of the chain. Wait until the routing tables have stabilized. Then kill node C and wait for the routing tables to stabilize. Capture the entire output of the three sessions using, for example, `script`.

4. Question: What happens after the node C fails and why? Describe the general progression of your output for nodes A and B in no more than three sentences.

5. Question: The periodic rules we have specified age routes and then send updates. Instead we could send updates and then age routes. Which scheme is better and why? Hint: consider the first test case (with a triangle topology). You may want to try changing your code to see the effect.

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

1. One C file called hw3.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 and sections that contains:

    a. A printout of the output from the test cases described above.

    b. A printout of the source code you submitted electronically.

    c. Short answers to the discussion questions above.

—END—