# Fishnet Assignment 2: Distance Vector Routing

**Out: Wednesday, Jan 23, 2002.**

**Due: Thursday, Feb 7, 2002 at 5 p.m.**

**CSE/EE461 Winter 2002; Anderson.**

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 first 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 called hw2.c that implements the Fishnet echo protocol and a distance vector routing protocol, as described below. Continue to bear in mind the Robustness Principle: "Be conservative in what you send and liberal in what you accept." This specification may leave some points ambiguous; do what you think is best as long as your program can interoperate with the sample solution and other nodes, and document the design decisions you make.

- Takes three command line arguments and joins the Fishnet described by those arguments. This is identical to assignment 1.

- 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 first assignment, and the output of your program for sending and receiving echo packets should also be identical in form to the first assignment.

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

- Performs a periodic update every 10 seconds. (Use the timer interface described in fish.h.) As part of the periodic update, call

  ```
  fish_debug(FISH_DEBUG_ROUTING, "Periodic update\n")
  ```

  to print a debugging message. Then age the information in the routing table and send routing updates to all neighbors, as described below.

- To age the routing table, you should remove any entries that have not been refreshed (as described further down) for the past three consecutive timer intervals.

- Then, use the `routing_packet` structure defined in fish.h to send a routing update to all neighbors. The value of the packet header's protocol field should be set to FISH_PROTOCOL_ROUTING; the destination should be ALL_NEIGHBORS. The packet 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.) Note that the size of a routing packet is `PACKET_HEADER_SIZE + num_adv*sizeof(struct route_advertisement)`.

- Receives routing update messages from neighboring nodes and uses them to update and refresh the contents of the routing table as follows. 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 C learned from preferred neighbor N.
- Every time a routing table entry is added, changed, or refreshed, its age should be set to 0.
- The distance you record in the routing table should always be one more than the distance advertised by your neighbor.
- When you receive a route for a new destination at a cost less than INFINITY, add it to the routing table.
- When you receive an advertisement for a route that is cheaper than the route currently in the routing table, replace the existing route with the new one.
- When you receive an advertisement for a route from the preferred neighbor and its cost has not changed, refresh the route by setting its age back to 0. If its cost has changed, update the routing table entry to reflect that change. If the cost of the route is now INFINITY, remove the route from the routing table.
- Prints the following messages using fish_debug when changing the contents of the routing table. The new route (or the route being deleted) is to destination D via preferred neighbor N at cost C.
- When adding a route to a new destination, print "Route add to D via N cost C".
- When changing a route to a known destination, print "Route change to D via N cost C".
- When refreshing a route, print "Route refresh to D via N cost C".
- When removing a route, print "Route remove to D via N cost C".
- Forwards any 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. If there is no route for the destination, print "`D unreachable`", where D is the destination.
- As before, your program must perform all tasks in any order and run until you give the

command "`exit`", when libfish. will end the program.

- You may choose to speed up route convergence by adding triggered updates. Whenever a received update changes your routing table, immediately send updates to your neighbors. Try this on the turnin test cases to see the difference.

- You may choose to speed up route convergence by adding the split horizon with poison reverse heuristic. With this heuristic, you advertise routes back to the neighbor you learned them from at cost INFINITY, rather than the cost in your routing table. This requires sending different advertisements to each of your neighbors. You can tell who your neighbors are by checking your routing table for routes where the preferred neighbor is the destination. You should continue to advertise yourself at cost 0 to ALL_NEIGHBORS, so new neighbors can learn about you. Try this on the turnin test case to see the difference.

## 1. Step-by-Step Development 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 hw1.c by copying it to the new file hw2.c. You may split the program's functionality between multiple source files; either use `#include` to directly include the other .c files in hw2.c, or modify the Makefile.

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 routes learned from advertisements 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 try joining the class network and doing the turnin cases.

## 1. Turn In and Discussion Questions

To prepare for turnin, you need to join the class network, gather the output of your program running the test cases below, and answer four discussion questions:

1. Describe distance vector routing in no more than three sentences.

2. 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 C (with a ^C) and wait for the routing tables to stabilize. Try sending from A to C and observe the result. Now restart node C, 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.

   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.

3. Run a three-node chain network by starting the fishhead with the argument "`–topology chain`" 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`.

   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.

4. Question: The implementation we have specified ages routes before sending 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. Join the class Fishnet at jimbo:7777 and try to keep this node running after your submission. Also, join the class network and send a message to node 1, the bulletin board node, as in assignment 1.

2. One or more C files containing the source code of your solution, and the Makefile if you modified it. The main file should be named hw2.c. Submit this electronically using the `turnin` program on the Linux servers.

3. One stapled paper writeup with your names and sections that contains:

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

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

c. Short answers to the discussion questions above.

—END—