Fishnet Assignment 2: Naming and Link-State Routing

Due: Friday, Oct 31, 2003 at the beginning of class. Out: Friday, Oct 17, 2003.

CSE/EE461 Winter 2003; Wetherall.

Your assignment is to extend your Fishnet node with a “DNS” naming service, so that you can ping hosts by their names, and a link-state routing protocol, so that messages are sent only to their destinations rather than being flooded. Be sure to download a new Fishnet distribution before you begin.

1 Naming

Your node must maintain a listing of all nodes that are currently alive in the Fishnet along with their hostname and Fishnet address. For example, this allows you to associate the node address 3 with the hostname “djw-ipaq.” Here are your constraints:

- You can use ONLY packets of type `struct name_packet`. These packets are designed for flooding information about naming entries throughout the network. Hostnames are strings, up to 11 characters and null-terminated, and selected by you to name your IPAQ. Empty strings should be taken to mean that the associated address is no longer alive in the Fishnet. The sequence number allows you to determine whether an entry is new; higher is newer. Each entry has its own sequence number set by the original sender for the entry so that you can carry multiple entries in one packet, should you want to do this. The valid interval is the number of milliseconds before the entry should be discarded as being too old and representing a node that is no longer reachable. It should be at least one minute to reduce the load on the network.

A good design will keep the hostname to address listings at every node as up-to-date as possible, yet use very little bandwidth. You will probably want to modify your ping command so that it takes a hostname to ping as well as an address. You will probably want to add a new “ls” command that lists the nodes in the network. To see that your protocol is working, you might use the fishhead to set up a small network and see that your “ls” command at one node tracks the fishnet membership as you stop and start nodes.

2 Link-State Routing

Your node must implement link-state routing and use these routes to forward packets. ***You should read about link-state routing in Peterson 4.2.3***. Note that we are not implementing OSPF, but a different and simpler link-state design. This generally involves four steps:

1. Neighbor discovery, to determine your current set of neighbors.
2. Link-state flooding, to tell all nodes about all neighbors.
3. Shortest-path calculation, to determine the next-hops for routes using Dijkstra’s algorithm.

4. Forwarding, to send packets using the next-hops.

Here are your constraints:

- You can ONLY use packets of type `struct neighbor_packet` to implement neighbor discovery. Conceptually, this is the same as using ping, but using a different type of packet produces a cleaner solution than overloading ping.
- You can ONLY use packets of type `struct link_state_packet` to distribute link-state information. These packets are designed for flooding and are very similar to the naming packets. Each entry describes a directed link from node A to node B and has a sequence number and validity interval. Each entry also has a cost for the link, which is either 1 (one) if the link is up, or INFINITY if the link is down.
- To run Dijkstra's algorithm, see the details in Peterson 4.2.3. Also note that you should not use the directed link AB in your database unless the directed link BA is also in your database and both have cost of less than INFINITY. The result of this algorithm should be a routing table containing the next-hop neighbor to send to for each destination address.
- You should forward packets using the next-hops neighbors in the above routing table (rather than the broadcast to ALL_NEIGHBORS and flooding from assignment one) except for the following packets: packets sent to the network broadcast address, which should be flooded as before; and packets of type `struct neighbor_packet`, which should only be used for neighbor discovery as described above.

Once you have completed all of the above, you should be able to ping any other node in the network, and have a packet travel to that node and back without being unnecessarily flooded throughout the network. To see that your protocol is working, you might set up a small ring network, use ping to test whether you can reach remote nodes, and then break reachability by stopping a node on the path so that ping no longer receives a response. Your routing protocol should detect this and repair the situation by finding an alternative path. When it does, ping will work again. This is the turn-in exercise.

Congratulations! You have a real, working network!

3 Discussion Questions

1. Why do we use a link for shortest paths only if it exists with cost less than INFINITY in our database in both directions? What would happen if we used a directed link AB in that direction regardless of whether there was a directed link BA with cost less than INFINITY?
2. Does your routing algorithm produce symmetric routes (that follow the same path from X to Y in reverse when going from Y to X)? Why or why not?
3. How could you improve your naming protocol if you were designing it from scratch?
4. How could you improve your routing protocol if you were designing it from scratch?

Write only a few, short sentences for each of these questions!

4 Turn-In

For this and future assignments, you need to demonstrate that your IPAQ works in the class network as well as turn in both electronic and paper material as follows.

1. Run a four node private fishnet (using your program only, not the sample solution), with each node running in a separate window. Make the nodes form a ring network (see `./fishhead --help` and look for `ring`). From one node, ping the other node that is not directly connected, observing which way it travels around the ring. Stop the intermediate node the messages passed through. Repeat the ping when routing has repaired paths and it travels along an alternate path. Capture the entire output 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 what is going on.
2. Use your IPAQ to join the Fishnet containing our node running in Sieg 324, the Systems' Lab. Make sure your node is exchanging neighbor, naming, and routing packets with our node. What is the name of our node?
3. Use the `turnin` program on the Linux servers to electronically submit one or more C files containing the source code of your solution. 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. The target must be called hw2, and you should turn in a Makefile if you changed the one 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.
    c. Short answers to the discussion questions.

—END—