# Fishnet Assignment 2: Routing

**Due Friday, October 29, 2004 at the beginning of class.**

**CSE/EE 461: Autumn 2004**

Your assignment is to extend your Fishnet node to support routing, so that messages are sent only to their destinations rather than being flooded throughout the network.

## 1 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 protocol. A link-state protocol 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 using Dijkstra's algorithm, to determine the next-hops for routes.
4. Forwarding, to send packets using the next-hops.

Here are your constraints:

- You should use the class LinkState in packet.rb to format link-state information. A LinkState object contains a list of neighbors for a given node. Note that the link state information is the "payload" (packet contents) of a packet with a normal packet header. The pack method of the LinkState class should be used to turn the LinkState object into a string before putting it into a packet that uses the Fishnet::LinkInfoPkt protocol.

- You should use your solution from assignment 1 to flood the link state information throughout the network, so that all nodes know which nodes are neighbors of which other nodes. A key design choice is when to send out a LinkState packet – periodically? Immediately after the neighbor list changes? A key design criterion is to distribute link state information using as few packets as possible, while keeping nodes as up to date as possible.

- To run Dijkstra's algorithm, see the details in Peterson. Note that you should not use a link in your routes unless both ends of the link agree that it is available (that they are neighbors). 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-hop neighbors in the above routing table. The exception is packets sent to the network broadcast address (e.g., for neighbor discovery or sending link state information); these should be flooded.

- You may want to add two commands to your node: "dump linkstate" to dump all of the link state advertisements you used to compute the routing table, and "dump table" to dump the contents of the routing table. You may find this more convenient than logging the entire routing table after every change.

- In a separate packet, but transmitted in the same way as the link state packet, you should transmit a short text name to more easily identify your node. The protocol

should be Fishnet::NamePkt; the packet contents should be the name of source node. For example, this allows you to associate the node address 16 with the hostname "brian-ipaq". You should also keep track of the names advertised by other nodes.

- You will probably want to modify your ping command so that it can take a hostname to ping instead of an address. Also, 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 simulator or trawler 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.

Once you have completed all of the above, you should be able to ping any other node (by name!) 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!

# 2 Efficient Routing (optional, for extra credit)

(Note that grades for the course are established without considering extra credit. We then add any extra credit points. Doing EC is entirely optional, and not doing them will in no way harm your grade, even if everyone else in the class does the EC problems. We offer EC problems as a way of providing challenges for those students with both the time and interest to pursue certain issues in more depth.)

Link state routing is a "mechanism" for controlling routes; it does not specify the "policy" used for selecting which routes to use and which ones to avoid. Clearly, the end user performance can vary dramatically based on the route selection. For example, what if certain wireless hops were very lossy? Shortest path routing might choose those routes, even though the end user might be happier with another path through more reliable links.

Part 2 is to design and implement a routing algorithm that chooses better routes than the default selected by link state shortest path. Better is of course in the eyes of the beholder -- lower loss rates, lower latency, higher bandwidth are all reasonable choices. Note that your design can and should make use of the facilities defined above (link state advertisements), but it need not interoperate with the code from other student projects (e.g., you can define extra protocol messages or carefully modify existing ones as needed). As a first step, for example, you'll probably want to implement something that measures the properties of a hop or a path, so that you can use that information in setting link weights for the shortest path computation or in choosing source routes for virtual circuits.

Be sure to add to your turn in: a short description of your design and a simple illustrative test case.

### 3 Discussion Questions

1. Why do we use a link for the shortest path computation only if it exists in our database in both directions? What would happen if we used a directed link AB when the link BA does not exist?
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. What would happen if a node advertised itself as having neighbors, but never forwarded packets? How might you modify your implementation to deal with this case?
4. What happens if link state packets are lost or corrupted?
5. What would happen if a node alternated between advertising and withdrawing a neighbor, every few milliseconds? How might you modify your implementation to deal with this case?

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 fishnet simulation or emulation. Make the nodes form a ring network. 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 that we can see what packets are being sent and received during the exchange of LinkState messages and as the ping goes across the network. Mark up the printout to tell us what is going on.

2. Use your IPAQ to join the Fishnet containing our node running in Allen 391, the Network Lab. Make sure your node is exchanging neighbor 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 Ruby 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 (e.g., if you decide to split your code into multiple files, make sure to load the other files in node.rb).

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 brief (probably less than a page) design document explaining your design decisions.
   c. A printout of any output we have asked you to capture.
   d. Short answers to the discussion questions.

5. Bring your iPAQ with your code on it for the in-class demo!

—END—