CPS372 Programming Project 2 – Link State Routing Simulation
Due: Wednesday 4/22/2009

Overview
In this programming assignment, you will create a simulation that performs link-state routing and packet forwarding in a packet-switched network. This simulated network supports only unicast (point-to-point) communication. The simulation will be driven by a sequence of timestamped events (sorted by increasing time) read from an input file and your code will write output messages to a separate output file. Events in the simulation consist of link-state routing messages and packet arrivals which in turn must be forwarded on to the appropriate interface. Whenever a link-state packet arrives, your code must update its data structures to maintain shortest path information to all hosts in the network. This shortest path information will then be used to build forwarding tables which will facilitate correct packet forwarding.

Link-state routing
Our protocol for link-state routing packets closely follows the description in our textbook. In particular, link-state packets (LSPs) in our simulation will consist of:

- the address of the node that generated the LSP
- a sequence number
- a list of pairs defining the distances to the nodes directly connected to the node which sent the LSP

The above packet is also referred to as link-state advertisement

The sequence number is used to differentiate new updates from stale updates. For every host that has transmitted an LSP, you should keep track of the largest sequence number that it has used so far. Arrival of an LSP from a host with a smaller or equal sequence number to the maximum seen so far from that host should be discarded. Sequence numbers for each host are unrelated to sequence numbers from other hosts. Also, do not worry about sequence number wraparound of LSPs in this assignment. *(In other words, my test data will be friendly - it will not generate a sequence number wraparound.)*

A special link-state packet will be used to initialize the simulation, and will consist of:

- the address of the router that you are to simulate
- a list of pairs defining the distances to the nodes directly connected to our router

At any point in time, given a view of the network topology (a list of connected routers and the cost to those routers), a router can run Dijkstra's algorithm to compute shortest paths from the router to all hosts. For the purposes of routing, it is not necessary to store the entire shortest path -- it suffices to run Dijsktra's and then store the first hop to every remote node by building a forwarding table similar to that presented in class and in our text. Pseudocode for Dijkstra's algorithm and a clear discussion of the issues in building forwarding tables from this information are in the text.

The question of how often to run Dijkstra's algorithm is an important one -- running the algorithm can be somewhat time-consuming, especially for large networks. I suggest running it only when necessary, i.e. only when both the state of the network has changed AND a new forwarding request has arrived. (*In other words, calculate the new forwarding table – only after receiving "new" link advertisements and when there is a packet to forward.*)

While most of the test cases will consider small networks of 5-50 nodes, your code should be designed to handle much larger simulations which can run into the thousands of nodes, and we will generate at least two test cases for large networks.

Some special cases may arise. In particular, at initialization you may find that you are node 71 and that you are directly connected to nodes 14, 17 and 81. Later, a node 95 (which you've never heard from) may report that it is directly connected to 71. This is valid, so make sure that your design can support additions and deletions to the set of nodes to which you are directly connected. Similarly, the weights on any link, including those directly connected to the router that you are simulating, may change over time. (In other words, a new link advertisement may report a change to a currently believed link cost.)

Addressing

Valid addresses of nodes in our network are positive integers between 0 and 2^15 - 1 inclusive.

Unicast forwarding

Forwarding packets traveling across point-to-point connections will be straightforward -- simply compute the next hop from the forwarding table then call the provided output routine: (if you plan to use the provided C code.)

void ForwardUnicastPacket (int time_now, int destination, int nexthop)

This will write the appropriate forwarding information into the output log.

Your program should be working completely correctly on event logs which consist only of the initialization sequence, link state packet arrivals, and requests to forward unicast packets.

Testing

To test your code, I will provide several event sequences for you to test your router on along with the correct output behavior. You should of course conduct additional test cases that you generate yourself. You have lots of time to work on this assignment, but please start early, and please make sure to allocate enough time to be able to submit a nicely debugged assignment.

Implementation and Submission

You are free to implement a solution to this assignment in any language you choose, but input parsing and output generation routines are provided in C only (it should be very easy to translate this to languages such as C++ or Java). As with project assignment 1, project assignment 2 should be submitted via the class dropbox. Please plan ahead to insure that you submit your solution to our class dropbox in a timely fashion.

What to Turn In

When you hand in your programming assignment, you should include:

- A source listing(s) containing in-line documentation. Uncommented code will be penalized.
- A separate (typed) document of a page or so describing the overall program design, a verbal description of ``how it works'', and design tradeoffs considered and made. Also describe possible improvements and extensions to your program (and sketch how they might be made).
- A separate description of the test cases you ran on your program to convince yourself (and me) that it is indeed correct, and execution traces showing these test being run. Also describe any cases for which your program is known not to work correctly. The system capabilities which your test cases should demonstrate are described below. (Feel free to share your project testcases with others in the class – perhaps these will assist you in straighten out your code.) [Note: several testcases will be provided by the instructor]
- A detailed description of how to build the program and execute the program.
- Place all of your documents/source files in a folder called name_project2 and drop into class dropbox – cps372dropbox

Grading

All programming efforts must be conducted individually and all code must be original. Assignments submitted on or before on 4/22/2009 will be graded out of 100 points. Assignments submitted on or before 4/24/2009 will be graded out of 80 points, assignments submitted later will not be graded.

Sample tracefiles and some basic helper routines (which you don't have to use) are provided. *Adapted from CS 455/655: Computer Networks, Prof. John Byers (Boston University)*