CS 438 Spring 2014 MP2

Due Friday March 14th, 9PM Central Time

Implementation of Unicast Routing Protocols

In this MP, you will implement link state and distance vector routing protocols. You may work with a partner. There is a 2% per hour late penalty. You will write three separate programs: a program to run a node in the link state protocol, a program to run a node in the distance vector protocol, and a manager program to convey information to the first two. The manager program is not protocol specific; your link state and distance vector programs will both receive information in the same format. We might test your programs with up to 16 simultaneous instantiations of the link state nodes, or of the distance vector nodes.

NOTE: We are calling the "names" that your link state/distance vector nodes use "virtual node IDs". These would be IP addresses in a real world routing protocol implementation. Since your programs will communicate over TCP/UDP/IP, and so use IP addresses, we're avoiding the word "address" when talking about the virtual topology to avoid confusion.

The Manager Program

The manager program reads a topology file and a message file (see below for format) and listens for connections from nodes. At any time, additional topology information can be entered on stdin, coming in the same format as in the topology file. The manager should update its topology accordingly.

Every new client that connects to the manager gets assigned the topology's lowest unclaimed virtual node ID, and is told about all of the neighbors it has in the virtual topology, along with those links' costs. This connection should stay open as long as the manager and node are alive. As new topology info becomes available, the manager should send it to all relevant nodes.

The nodes also need to know how to talk to their virtual neighbors. Therefore, when a node first contacts the manager, along with its virtual node ID and list of neighbors, it should be told the IP addresses of any neighbors who have already contacted the manager. For neighbors who join later, the manager should send the IP addresses once they're available.

After the manager hears that the nodes have converged, it will tell certain nodes to send messages to certain others; more details below.

Router Programs – Non-protocol-specific parts

Nodes should only directly communicate with the manager, and the nodes that the manager has told them are currently their neighbors in the topology. This is the only restriction on how your nodes communicate with each other; they need only follow the concepts of link state/distance vector, and not send vastly more control traffic than an efficient implementation would. A node should connect to the manager on startup, having

taken the manager's hostname or IP address as its only command line argument.

Your nodes' communications can be purely reactive, rather than periodic: you only need to send an update to your neighbors when your forwarding table (for distance vector) or view of the topology (for link state) changes, rather than "whenever there's a change, or if it's been x seconds since the last update".

Your nodes should have a procedure for determining that everyone's tables have converged. This could be as simple as waiting until several seconds have passed without any changes, or could be an actual consensus protocol. It's up to you. The manager should be informed of the convergence once the nodes think it has happened.

Once the tables have converged, your manager will instruct some of your nodes to send some data to some other nodes, with the data forwarded according to the nodes' forwarding tables. The sources and destinations are specified in the manager's message file; see below for format. For more convenient output, every node in the path will append its virtual node ID to a list included in the packet when it forwards it. So, if a packet is going to be sent 3->5->1->2, this list should read 351 when the packet reaches node 2.

After the data is sent, nothing else needs to happen until the manager tells the nodes about a topology change that was entered on stdin. The nodes behave according to their routing protocol. Once the manager believes the network has reconverged from this change, it instructs the same senders to send to the same receivers. The manager then goes back to waiting for a topology change from stdin.

Router Programs – Protocol-specific parts

Given the virtual topology provided by the manager, your distance vector nodes should use the distance vector algorithm to arrive at a correct forwarding table for the network they're in, and your link state nodes should use link state. Remember, a node can only exchange messages with its neighbors in the virtual topology. We recommend UDP for the node-to-node communication; your code will probably be cleaner that way.

Input Formats

The manager reads two files: the topology file and the message file. Both files have their items delimited by newlines. The topology file represents, shockingly, the topology. The message file describes which hosts should send data to whom once the tables converge.

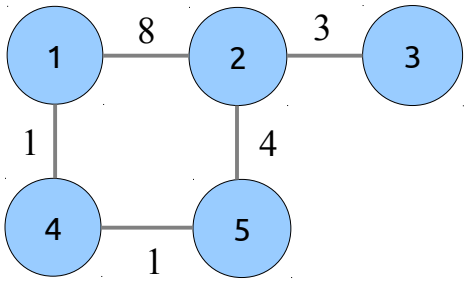
A line in the topology file represents a link between two nodes, and looks like:
<ID of a node> <ID of another node> <cost of the link between them>
This same format can be entered to the manager's stdin to update the topology. Cost < 0 indicates that the previously existing link between the two nodes is broken.

A line in the message file looks like:
<source node ID> <dest node ID> <message text>
The files we test your code with will follow this description exactly, with nothing extraneous or improperly formatted.

Example topology file:
1 2 8
2 3 3
2 5 4
4 1 1
4 5 1

Example message file:
2 1 here is a message from 2 to 1
3 5 this one gets sent from 3 to 5!

This would correspond to a topology of:



The message file would cause "here is a message from 2 to 1" to be sent from 2->1, and "this one gets sent from 3 to 5!" from 3->5.

Output Format

Your manager program does not need output. Your nodes must print exactly the following on stdout:

- "now linked to node x with cost y" whenever the manager tells them they are linked to node x with cost y

- "no longer linked to node x" whenever the manager tells them that their link to x has been broken

- When a node sends, forwards, or receives a data message, it must print the message, the source + destination, and the list of nodes visited (after appending itself to that list) in this format: “from <x> to <y> hops <hop1> <hop2> <...> message <message>“ e.g. node 4 might print “from 2 to 1 hops 2 5 4 message hello there!”

Distance Vector only: print the forwarding table when and only when it *converges* to *something new*. The format should be:
destination nexthop pathcost
Where nexthop is the neighbor we hand destination's packets to, and pathcost is the total cost of this path to destination. The table should be sorted by destination. Example for node 2 from the example topology:
1 5 6
2 2 0
3 3 3
4 5 5
5 5 4
As you can see, the node's entry for itself should list the nexthop as itself, and the cost as 0.

Link State only: print the routing table when and only when it *converges* to *something new*. The format should be:
destination pathcost: hop1 hop2 hop3 hop4...
Where each hop is a hop on the shortest path. (Hop1 should always be the source node itself, and the last hop should be the destination). Sort by destination. Example for node 2:

1 6: 2 5 4 1
2 0: 2
3 3: 2 3
4 5: 2 5 4
5 4: 2 5

Please do not print anything else; any diagnostic messages or the like should be commented out before submission. However, if you want to organize the output a little, it's ok to print as many blank lines as you want in between lines of output.

Notes

- **IMPORTANT**: You must use C or C++, and we will not accept submissions that can only run in the context of an IDE. Running your project shouldn't involve more than "make" and invoking the executables.

- You can work with one partner or no partner. We suggest that one partner takes distance vector, the other takes link state, and both work in conjunction on the manager. **Put your and your partner's names and netids in a README!** Only one partner needs to submit a copy. If you worked alone, put your name and netid in the README, along with an indication that you worked alone. (You must still implement both protocols if you work alone).

- Your project must include a Makefile whose "all" target makes executables called manager, distvec, and linkstate.

- The manager must run as "./manager topologyfile messagefile".

- distvec and linkstate must run like "./distvec managerhostname".

- Portability shouldn't be an issue, but if your code works on EWS, that's good enough.

- Submit MP2 like MP1: in a directory called MP2 committed into your CS438 svn directory. To be sure it's committed: cd into the MP2 directory, and run "svn ci ; svn list". Any file listed is on the server and up-to-date.

- If you use UDP, don't worry about reliability. We'll test in an environment unlikely to drop packets, unless your protocol sends WAY too many messages.

- **Hint#1**: if you do your testing on a single computer, your nodes will all have the same IP address, and so will need to use ports to distinguish themselves. Try having nodes listen on UDP port <fixed number>+<their virtual node ID>.

- **Hint#2:** with hint#1, if you sendto() on the same socket you recvfrom() on (must be bound), the source port of your sent packets will be the port you're listening on.

- Your code should be yours and your partner's. Issues will be handled in accordance with the UIUC Academic Honesty and Student Conduct Policies.