Sabir Kirpal
Oct 10, 2021
CSE 160
Project 2

Discussion Questions.

1. If you used a directed link for computation, packets would be able to be sent out but acknowledgments of receiving these packets would not be possible if you can't get back from B to A
2. My routing algorithm follows symmetric routes only due to the fact that I designed an Adjacency Matrix for our network topology. In a undirected weighted graph, because both sides of the diagonal 0's are symmetrical, similar shortest paths will be found in both directions.
3. If a node advertises that it has neighbors but doesn't forward that packet, then no other node in our database will know this nodes neighbors even though it has a direct path. In this case we might want to modify our code to handle asking for its neighbors again before computing Dijkstra's.
4. If link state packets are lost or corrupted, our database will no longer be updated and will either take a suboptimal route or not be able to reach to destination.
5. If a node advertises and then withdraws a neighbor we similarly need to re check for any updates before computing Shortest Paths again before sending out any message or forwarding any message.

Design Writeup

In order to implement Link State algorithm, only one more interface was introduced continuing on from project 1. This is the LinkStateP.nc file which holds my implementation. Initially the main problem was how to keep an updated database for all nodes if we can't just send directly to other nodes. To get around this problem I implemented a similar mechanism to flooding where each node updates that packets payload into its data base and then forwards the packets to all it's neighbors using AM_BROADCAST_ADDR.

Another key design issue before I started coding was when to compute shortest paths? At first I implemented a timer that would fire much slower than our lsp discovery timer however that was very inefficient and it would only work if you changed TestSim.py to ping at 3-4 seconds in. I instead thought to calculate these shortest paths whenever we would like to ping somewhere or forward a packet. This is done because we know it is unlikely for someone to ping before they finish finding their neighbors at least once, and if that does happen it would most likely just generate a suboptimal route which is fine for our purposes (that is the best we can do)

LStimer :

- advertises Link State Packets by updating TOS_NODE_ID's (its own) database row and then floods same advertisement everywhere for others to update that nodes row in their database.
- advertiseLSP();

ComputeSP() :

- reads database and uses info to calculate optimal paths for each node in the network
- Dijkstra's was used through a queuing system
    - SP - holds all considered nodes
    - Computing - holds all in progress nodes
- Uses popMinimum and appendNode functions for queues

HandlePacket():

- this command is used in flooding to route the packets to a designated event handler
- flooding packets go somewhere else, neighbor discovery packets go somewhere else, lsp packets go somewhere else
- handlePacket checks to make sure we haven't already seen this packet here before
- already have checked it TTL is expired in flooding
- checks if at destination -> reads it and computes shortest path again
- other wise compute shortest path and forward to next hop neighbor.