UCMerced Fall 2024 - CSE160 - Computer Networks
Project 2 - Link State Routing
Project Team: CJ Cox, Aaron Nam
Professor Cerpa

Design Process & Decisions

In Project 2, We modularize the routing algorithm to our data structures and then create a module and interface for Link State. The reason for doing this is for high-level design to eventually incorporate the IP module portion for Project 3. Before implementing Link State and Routing, we had to improve upon our flooding and neighbor discovery. After we linked up the modules of neighbor discovery and flooding, we designed neighbor discovery to check the connection strength of each node. This decision ensures that the system can detect if nodes are in range and are a strong enough reliable communication.

When we implemented the routing algorithm, we decided to use Dijkstra as our best option. It was chosen because of its efficiency in calculating the shortest path between our nodes. This helps since we made our edges at a value of one to each node, for example, if A→B or B→A, it is practical for symmetric routes. This was essential because, in our flooding-based network, we want to optimize and minimize the number of hops for each transmission. It overall improves the performance and efficiency of our network.

We used the hashmap to store all the updated neighbors and made a command called “getNeighbors().” This function would then be called to FloodP.nc, the function “call NeighborDiscovery.getNeighbors(), which gets the list of neighboring nodes and connection strengths. It is stored in updatedNeighbors which is the current state of the node’s connections. We want to continuously monitor the status of the network by comparing it to a previously stored list as it is shown in our debugger. We made a design where once the package reaches its destination it will provide the node the “neighborGraph” which allows the node to know its immediate neighbors and the source of the node the package was originally sent from.

After checking the stability of the neighbor list, we made the process of flooding the payload, so when we send the payload the neighboring nodes will flood the message to each other until the payload is received. The decision ensures that the message will propagate through the network so we can detect new nodes in the case of a topology change. In the case that the topology changes, the network will be considered temporarily unstable until the neighbors' list is stable in multiple cycles. As long as the neighbors' list is stable it will proceed to flood the messages so that the payload gets to its intended node (or router).

UCMerced Fall 2024 - CSE160 - Computer Networks
Project 2 - Link State Routing
Project Team: CJ Cox, Aaron Nam
Professor Cerpa

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?

A: In directed links going bidirectional, the idea for using shortest path computation is to have a two-way communication so that it the routing doesn't loop. For our project, at the moment only one-way links were implemented in our Link State, each node runs individually through Dijkstra. So the links only set one direction at the moment.

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?

A: Our routing algorithm does produce symmetric routes since the link costs in the graph are in both directions all edges have a cost of 1, we also implemented Dijkstra so it always chooses the shortest path. In the example of X to Y and Y to X, it will be the same in reverse.

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?

A: We created a flood reply mechanism that helps detect non-forwarding nodes, it has a two-way communication. The non-forwarding nodes won't relay the replies, so it won't be included in the routing algorithm. For example, If the topology is linear and it has nodes on the left and right, it would never see those neighbors. If there was a node that did an advertisement on itself it would be ignored currently in this implementation since we are always forwarding packets through the neighboring lists already discovered (with full connection strength provided).

4. What happens if link state packets are lost or corrupted?

A: If the link state packets are lost or corrupted then the information will likely just refresh periodically every 10 seconds in our code.

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?

A: From question #4, our refresh period, so in the situation given our network it would accept neighbor changes. To modify our implementation we would need to check stability and have

some sort of stability window to see whether it is stable, if it is then we can ignore neighbors that change frequently.