Marcus Loe & Byron Garcia
Computer Networks
10/10/18

Project 2: Distance Vector Routing

Discussion Questions

1. **What are the pros and cons of using distance vector routing compared to link state routing?**

Some of the benefits of vector distance routing are that it takes minimal configuration to set up compared to link state routing which is more complicated. Yet, unfortunately distance vector routing is costly and slow. It is prone to routing loops like the count to infinity problem, unless the optimizations like split horizon and poison reverse are in place. Link state already has these optimizations in place to prevent routing loops. Another negative of distance vector routing is the slow convergence due to ripples and hold down. Link state routing in comparison ha faster convergence, but higher costs due to the flooding.

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?**

The routing algorithm should produce symmetric routes since the algorithm looks for the shortest path between nodes, so the path from X to Y should be the shortest path from Y to X.

3. **What if a node advertised itself as having a route to some nodes, but never forwards packets to those nodes? Is there anything you can do in your implementation to deal with this case?**

Our implementation has a solution for that. We first check if the node has a reply, and the cost count is incremented. When the node doesn't forward the packet to the routes that it advertises to, the route and node gets dropped from the routing table.

4. **What happens if a distance vector packet is lost or corrupted?**

If a packet is lost then an update of the packets happens, but if a corrupted packet is sent with corrupted information about routes then poison reverse kicks in to prevent a potential loop.

5. **What would happen if a node alternated between advertising and withdrawing a route to a node every few milliseconds? How might you modify your implementation to deal with this case?**

   If the node alternated between advertising and withdrawing a node, there would be too many collisions for any packets to be efficiently and properly routed. The routing table would have issues with keeping track of what nodes are available to have packets sent to it. Our implementation could be modified to force an "advertised" uptime where the node has to be active for x amount of seconds in order for a packet to be sent to it.

Write up

Distance Vector Routing

In order to discover the neighbors directly around us, we used timers set in milliseconds to shoot out a command to the nodes next door. As given, am_broadcast_addr ensured all the neighboring nodes gets a command and returns a reply that contains the identifier into a list. However, upon thinking further, a hashmap should've been implemented to have made future projects search the neighbors faster. If the network has seen the neighbor or packet before, it wouldn't need to search through the list again. We then send a packet to the neighboring nodes and ensure a reply is sent back.

The list containing the identifiers will continually check if the neighbor still exists. Within 3 or so call's if the neighbor does not respond, it is dropped from the list. However, dealing with packets was a little more tricky. Another way we experimented with was an array. However, after implementing it we discovered that arrays required manual configuration size, and without a definite way to dynamically size the array, the network would only run on the TOSSIM, and not on any real world scenarios.

In order to determine if packets were alive for too long, we had to implement something. Using timeouts was also key in determining if the packet has circulated for far too long. In addition, this "TTL" keeps the network clean and free of useless information. The packets are then circulated around the neighbors that are directly surrounding us, and a ping reply must be returned in order for that node to be added to the routing table. Instead of flooding the network for neighbors, only those nodes that have replied back properly and have a valid path to other nodes are included. The routing table holds all the nodes that reply, and creates a valid map of what nodes can have a packet forwarded to it. If the reply isn't returned by the node, the table is updated and the node is dropped.

As told previously, building project 1 properly in order for future projects is key. From a single Node.nc file we've modularized the program to allow additions to be easily made, and of course version control. In addition, we realized that nesC doesn't have the same functions as C, so we were limited by the scope of what we could do.