CSE 160
Computer Networks

Project #2 Report

October 11, 2018

Members

Chris DeSoto
Khushpreet Buttar

1 Design Decisions

Our design process for Project #2 began with an analysis of the project requirements and how the existing facilities built in Project #1 could be used to satisfy them. Our neighbor discovery implementation was, for the most part, suitable for the task of distance vector routing. One important change made the the neighbor discovery component was to modify the timing system that determined when a neighbor had left the network. Originally we had decided to use the node relative time to track how long it had been since the last contact with the neighbor node. This, however, was not the most ideal approach as it was both, overly complex and not wholly accurate. With these details in mind, we decided to replace the time with a simple unsigned integer count-down. This value is assigned to the neighbor hashmap structure each time a neighbor discovery response comes in. It is decremented every time a neighbor discovery request goes out. We decided on a value of 5 for this neighbor discovery TTL.

The first portion of the distance vector routing component was setup the timing component that is used to perform periodic updates and advertisements on the network. As recommended in the project specification we decided to give neighbor discovery an initial window to acquire knowledge of the network before attempting to send out distance vector packets.

Next was the task of creating the routing table structure. We decided to follow the implementation in the book and create a struct array. Each route struct contains a destination, a next hop, a cost and a TTL. To occupy as little space as possible, these values are all of type uint8_t. It is unlikely that more nodes will be supported because of the limitations inherent to distance vector routing. Routes within our table are entered in the order they are received. Up to 256 routes can be held in this table (uint8_t allows indexing of 255 nodes, excluding 0). Though it may have been simpler to index each route with the destination as the values of each node id correlate to the n-1$^{\text{th}}$ index of an array of length 256, our implementation allows larger node ids to be entered within a smaller table. This is more similar to a real world scenario. Though a node would be required to include a route to all nodes currently available within the network, not all nodes may be active within the network at the same time. If space were an important consideration, the route table size could be limited and a cap on the number of route-able nodes would be placed on each node. If not, it can simply be of length 256.

For the delivery of routes via packet transmission, we decided to maximize efficiency. Since the packet payload allows a maximum of 20 8-bit unsigned integers and the size of our route struct was 4 bytes, we could send a maximum of 5 routes per packet. This required a separate array of route structs to pass to the makePack function to be copied onto the packet payload.

The distance vector algorithm was fairly straight forward. The most subtle part of designing the algorithm was doing so in a way that would preserve the ability to properly update the TTL values for the routes in the table. Many conditions were checked in order to determine if the route received was a new route, an update of a current route, or a less optimal route. The TTL was set to refresh in all but the last case.

2 Discussion Questions

1. Some advantages of distance vector routing over link state routing include simplicity and low computational overhead. The distance vector algorithm is simpler than it's link state alternative. Because of this, it is much easier to configure and maintain. It is also computationally light as only simple comparisons of cost are made rather than complex algorithms such as Djikstra's. Some disadvantages include slower convergence, susceptibility to loops, larger routing tables. Because distance vector calculations need to be done before the routing table is passed on to a node's neighbors, convergence is slower than in link state routing. Though the count-to-infinity problem and be ameliorated with techniques such as split horizon, poison reverse, and triggered updates, it cannot be eliminated completely. Large routing tables are required to implement distance vector routing, because each node must have an entry in it's table for every other node, as well as, the next hop and cost for each. These tables also have to be transmitted periodically across the network to each node's neighbors eating up bandwidth.

2. No, symmetric routes cannot be guaranteed. Because there may be multiple routes which result in an equidistant path to a node, the entry in a particular node's routing table will simply be the route "seen" first. Therefore, one node could through a particular route and receive a reply from a different route of equal distance.

3. That could mean the advertising node does not actually have the route to the nodes, and it actually does try to forwards packets. If the advertised node does not receive any packets for certain amount a time, the advertised node can drop the connection. When neighbor discovery runs, it should trigger an event to no longer advertise the route.

4. By using TTL for each of the routes we can ensure that corrupted routes have a short lifetime. Because the corrupted form of the distance vector packet is unlikely to be seen again, the route's TTL will be continuously decremented until it is removed completely. Unfortunately, during that time the route will be advertised and any packets to that destination will be routed incorrectly. In the case of a lost packet, the consequences are less dramatic. Either the route will not be advertised (because it was never received) or the route's TTL will fail to be updated on the node set to receive the packet. These errors should soon be rectified when the transmitting node's distance vector timer is fired again.

5. Because of the nature of triggered updates, changes in advertisement every few milliseconds would cause a great deal of traffic on the network as it struggles to converge. To account for this holddown timers can be implemented within our distance vector routing algorithm. This technique sets a timer when an update for a particular route is received. Until this timer expires, no new updates for that route will be accepted. This would effectively dampen the alternating advertisement at the cost of prolonged convergence.