Marcus Loe and Byron Garcia

CSE 160: Project 1

9/20/18

Discussion Questions

1. Describe a pro and a con of using event driven programing.

Event driven programming permits nodes in a system to be independent without specific code written to link groups in a system. So a node could push out a packet without relying on a function on another node to trigger the packet to be processed, it can do it independently. Despite these benefits, event driven programming makes the code of the system more difficult to understand. It won’t be as clear if a node is reacting to something another node might be doing.

1. Flooding includes a mechanism to prevent packets from circulating indefinitely, and the TTL field provides another mechanism. What is the benefit of having both? What would happen if we only had flooding checks? What would happen if we had only TTL checks?

The benefit of having both flooding checks and TTL is to prevent packets from looping endlessly. Both are utilized in conjunction to work within big and small networks, flooding works best in small networks while TTL works best in big networks. If only flooding checks were utilized, it’d work quickly in small networks but slow immensely since it has to visit each node twice before the packet disappears. If only TTL were utilized the opposite would happen, it’d work well with larger networks, but not in smaller ones since the packet would be unnecessarily flooded to the small number of nodes. Having both allows to have both of their benefits, which can work in both smaller and larger networks.

1. When using the flooding protocol, what would be the total number of packets sent/received by all the nodes in the best case situation? Worse case situation? Explain the topology and the reasoning behind each case.

The best case situation has one packet being sent out and only one being received. In project 1’s topology, for example, node 1 could send a message to node 2. Node 1 would send a packet to its neighbor, node 2. Since the packet was intended to reach node 2, it would no go any further to no other nodes. In the worst case, alternatively, the number of packets received/sent is equivalent to the number of nodes. Assuming we are referring to the topology of project 1, the worst case would be 17. For this to happen each node would need to flood its adjacent neighbor, meaning all nodes would receive the packet eventually.

1. Using the information gathered from neighbor discovery, what would be a better way of accomplishing multi-hop communication?

With the information from neighbor discovery we can create a shortest distance algorithm if nodes tell other nodes its own neighbors. This will allow for multi-hop communication to become much more efficient within the network, since we know where to go when sending a packet, instead of flooding each node. It would also lower the amount of packet transfer in the system, but raise the amount of data used within a node.

1. Describe a design decision you could have made differently given that you can change the provided skeleton code and the pros and cons compared to the decision you made?

A change that could have been made in the skeleton code could’ve been in the CommandHandler.ping() function to have any ping packet sent to AM\_BROADCAST\_ADDR. It would simplify perform flooding by allowing any node that receives that ping packet to immediately send it back with little code telling it to do that. Unfortunately, this approach limits its versatility.

Write up

Neighbor Discovery/Flooding

In order to discover the neighbors 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.

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. As said above, a hashmap would’ve been more beneficial when going through the list to see if the neighbor exists. It would’ve had a O(1) runtime, an extremely beneficial tool when running networks. However, dealing with packets was a little more tricky.

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.

As told previously, building project 1 properly in order for future projects is key. Future plans include modularizing 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.