Renee Helfert
John Villalvazo
CSE 160
Project 1
Due: 9/22/2020 @3pm

Discussion Questions

1. Describe a pro and a con of using event-driven programming.

PRO: Event-driven programs are easy to alter and adjust since "events" in the code are compartmentalized.

CON: However, this can make the code cumbersome to read and analyze due to the number of files that a program can have for each event.

2. 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 is that it allows for a more narrow check to keep flooding issues to a minimum. Only having Flooding Checks might keep the flooding from happening longer than needed (i.e indefinite) and causing loops since not checking the TTL of a package might allow it to recirculate. Only having TTL Checks might stop the full flooding short and "hide" some nodes who were not reached by the flooding before it ended.

3. 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.

When using the flooding protocol, it is expected that the total number of packets to be sent/received by all nodes would be from 1 packet to (packets = # of neighbor nodes) if there were a loop somewhere within the topology. In this best-case scenario, the reason this would be ideal is that, in a given topology, for every node, there would be at least 1 packet addressed to it. No duplicates would reach any nodes within a topology, thus being the assumption of a correct implementation of the flooding protocol. The worst-case scenario would be something like n + 1 packets (where n = the number of neighbors of a given node). This can mean that there is a recycling of packets while the system is flooding which is not ideal. This could also indicate that within a given topology somewhere within the network a node would have a connection to another node that was not wanted/intended. Thus, the final node in a network, say a receiving

node, would end up receiving n+1 packets whereas only n packets would be sent through the network.

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

As of this time, we are not entirely sure what other technique would be more efficient, though we do believe that something similar to how a switch collects addresses based on when and where a package is sent to link up different nodes could be a viable substitution. This would mean nodes would only be discovered if they are being used.

5. 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 design decision we could have done differently was our method of storing the neighbors and package (sequences and sources) of a given node. We were thinking of implementing a list system from the data Structures file, but our lack of knowledge when it comes to the inner workings of nesC made the idea too difficult to implement in the given project time. Instead, we opted to directly initiate arrays to hold the data. While initializing arrays was much simpler compared to a list, it means that the amount of information each node can save is limited to how long the array we initialized is. Although this might not be a bad thing as long as the number of neighbors and packages does not regularly exceed the storage capacity of each node. Future additions to our project will use the now known knowledge of modules as well as syntax of nesC to further modularize and clean up our project.