Adam Pluguez CSE160:Project 1 Discussion Questions 2/12/19

Event driven programming refers to a programing paradigm in which program flow is determined by events. In a wireless sensor network an event driven system is very effective, as conservation of computation and power management are vital to network functionality. Without an event driven system nodes would need to continuously poll the network to detect and react to events increasing overall power use and decreasing performance. One con to event driven programming is that it is more complicated to code and debug due to the event based environment it is used in, often relying on hardware or detection in the environment for event triggers.

Having both TTL and flooding checks allows us to be sure that packets are not circulating through the network indefinitely. If we only used flooding checks this would prevent nodes from rebroadcasting packets they have already broadcast, but packets would still bounce through every node before being dropped. This would clearly lead to unnecessary traffic. Only using TTL would cause many unnecessary packets to be rebroadcast as nodes would not know what packets they have already forwarded. These packets would bounce through the network until the TTL timed out.

Given (n) nodes the worst case will take (n-1) nodes to reach the destination node. Being there are (n) nodes the worst case is then O(n^2). Following similar logic the best case would be that the message will only have to make one hop to its destination node. As there are still (n) nodes the total would clearly be O(n). Topologies that create these cases include, a line topology, where each node is connected to two other nodes excluding the first and last node. This means for the first node to send to the last it must pass through (n-1) nodes. A tree topology would cause the same effect if the root is trying to send to a outermost leaf, meaning for the root node to send to the leaf it must pass through (n-1) nodes..

By using known neighbor information nodes could forward packets in the direction of the destination, rather than broadcasting the packets to all neighbor nodes blindly. This would further cut down on unnecessary packets circulating through the network.

All coding was done inside Node.nc. Given more time and a better understanding of NesC code flooding and neighbor discovery are likely better implemented in their own configuration/modules. These then would be wired to Node for use. But overall I feel like I do not have thorough enough understanding of the code to make clear judgements about what would be improvements or not.

Adam Pluguez CSE160 Project 1:Code Design 2/12/19

Coding was done mainly in Node.nc, with some minor adjustments in NodeC.nc and TestSim.py. The first things to attempt was flooding, this required a list to keep track of what packets each node had already forwarded so there was no repeated traffic. A list was wired in from the data structures and a list of the data type pack was created to hold packets. If and else conditions were then set up in the event receiver to forward packets that had not reached their intended destination. This was done by first broadcasting a packet from the node one to node nineteen. The data in the packet was the original sending nodes ID, The packet is then forwarded by each node until reaching node nineteen updating us of its progress along the way. Upon reaching node nineteen the payload data showing the original source is displayed along with the sequence number showing that the packet was forwarded from node one to node nineteen through flooding.

Neighbor discovery was more complicated to sort out the programming logic for. The basic approach I used was to broadcast to all neighbor nodes which would then respond to the broadcast node causing that node to be added to the neighbor list for the broadcasting node. That second node would then broadcast to the next node repeating the process until all neighbors are recorded in the list as they exist topologically in the simulated sensor network. A timer is used to trigger the creation of the neighbor list through the function createNeighborsList. Also printNeighbors was modified to print the neighbors of the node it was called with. I am doubting this is completely correct implementation as I am not sure it would handle nodes disappearing correctly, given more time I could have corrected this. My condition unfortunately only allows me to complete so much work in a given time and this is split between three classes this semester.