Morgan Trembley

Project 1 Discussion

1. Event driven programming can be more interactive. As in the name, when an event happens somewhere, it triggers a response from the necessary interfaces. It is also more dynamic since resources are only uses when an event happens. Once the event is over, the resources are available for other events/purposes. Event driven programming can also be complex and uneasy to follow or understand. It can also be less predictable and harder to debug because of that complexity.
2. TTL can provide additional functionality since you can end a flood before the packet reaches all nodes. If only the sequence check was in place, flood would always reach the entire network. There may also be some circumstances where the sequence fails to identify the packet has already reached a node so a failsafe that can also provide a service is ideal. If we only used flooding checks the entire network would always be flooded. If we only had TTL there would be a lot of repeated sending in some cases until the packet died.
3. Best case, each node has 2 neighbors except the ends, which only have 1. In that case there would be n-1 sent packets and n-1 received packets. This is because each node would only have 1 valid neighbor to send to (except the last node) since the other neighbor would be the source of the packet and the protocol wouldn't send it back. Worst case, all nodes are connected to each other. Since a node only knows not to send it back to the source all nodes would send to all other nodes, wasting a lot of time and tasks. The sent/received packets would be on the order of n^2 in this case.
4. You could build a graph using the neighbor table's neighbor information. Then you can use an algorithm like depth first search to find a "route" to the destination, then send the packet through that route of nodes to reach the destination without wasting a bunch of time and processing sending it to every node when only 1 needs the payload.
5. There are some redundancies and unneeded processes that were implemented because some of the tasks were unclear to me during the building of the interfaces. There are some less than ideal implementations that tend to happen on a first draft of a program. I used some data structures that were easier to implement but take up more space, especially in the long line topology. A lot of the information being passed back and forth, and processes that could be skipped are also probably deficient in some ways. I would also wager that the interfaces could be fragile, and fail under special circumstances I did not think of or encounter while creating them. Using a hash table instead of my table approach to store and retrieve node/neighbor data is a much better option but would have taken more time to implement correctly and I'd probably have to change a large portion of the logic for printing and identifying neighbors if I were to implement it. Most of the code I put together was a prototype to try and understand the problem, skeleton code, and learn how to wire everything/format my code to work. The Pro's of what I did are mostly that it works, at least what I've been able to test for, and it was completed on time. I wouldn't have had time to make the improvements I would like to before the due date but plan to do so in the future.