Kaila Thornton
CS455 – Assignment 1 Written
2/22/2014

Q1) Assignment 1 presented various challenges along the way. While unfamiliarity with the concepts relating to networking was among the greatest, design choices and their relations to the overall program was the most difficult. I found myself constantly going back to classes I thought I had finished writing, such as some of the messaging classes, to change data structures that was being used. This also was true for data structures within my node classes. I never created a solid design plan in the beginning of the assignment, which proved to be a mistake. As I was writing the program, I had to take critical time away from solving more of the problem to simple go back and change design implementations. For instance, initially, I thought of storing my connections between the nodes within an array of connection objects. However, as I progressed through the problem, my conceptual view of what was going on in the system changed, and I realized the current data structures I was using were not the best choices for getting the job done. Eventually, my connection storage morphed into an ArrayList, and finally into a HashMap in order to support the functions I needed access to. I found it important in the end to be able to search for a connection based on the ID of the node/socket it was related to (host and port name). Whenever I was utilizing Arrays and ArrayLists, this would have been possible to implement, however, converting the structure to a HashMap made more sense as the Java language provides these operations easily.

The other greatest challenge associated with this assignment was underestimating the problem. Due to other classes also having programs due with shorter time periods to work on those, I ended up procrastinating this assignment a bit even though I knew it would be a beast. This turned out to be a mistake; had I at least tackled more of the design aspect of the assignment instead of immediately jumping into coding once I started working on this, it may not have proven to be so bad. However, my technique of "just code" and a late start led to a week of late nights in the computer science building and a still incomplete program.

Q2) Given the opportunity to redesign my implementation, I would go back and change a great deal. Due to my lack of design process from the beginning, I feel like my program has very little organization. Even though I followed the class breakdown outlined by Shrideep, I feel like my code lacks cohesion and organization. Much of it seems to be hacked together instead of deeply thought out. As I was programming, I would make decisions on how to implement various parts of the program on the fly, leading to very little thought going into how it would all work together. This made debugging a

bit of a mess and following my code smoothly to be difficult. I also would change my hierarchical structure a little bit. Though I did utilize interfaces to ensure certain methods would be present within different sections of the program, I believe a little more inheritance would have made everything feel much more like one final product. There were methods that I found myself rewriting in different classes that were essentially the same, particularly for my wire formats. If I had used an overarching "message" parent class, there would have been certain variables and methods common to all that would have been easier to implement and much more efficient. Efficiency in my code is another aspect I would go back to reconsider. While I was programming, I paid no attention to the code efficiency because I simply wanted to finish in time to get some points for it. Unfortunately, now looking back through some of my code, I believe that there are portions of it that are needlessly slow and could very easily be redesigned into something better. For example, as the program computes the different overlay connections and link weights for the nodes, I am essentially iterating through a HashMap multiple times when I could have combined actions to make it more efficient.

Q3) Given a situation in which link weights are changing constantly, the shortest path algorithm would have to nearly constantly be running. In order to route a message efficiently so that it follows the shortest path it can, at each node, the shortest path would have to be recalculated. The packets/messages would have to be aware of the overarching overlay structure and knowledgeable of what paths to the final destination existed. If this information was included in the messages, Dijkstra's would need to be ran every time a packet was received. Since the shortest path algorithm would be ran at every node, the packet would only have to worry about the beginning node on the shortest path to their destination. It would not matter to the packet what the full path is, because it is very likely that this would change as it moved along throughout the system. This form of an implementation would have some issues of its own that would have to be dealt with, as it seems like packets could easily just be wandering around different nodes without ever reaching the final destination, or taking quite a long time to reach the final destination.

Q4) No, it is not necessary to create a specific routing plan per message. Keeping track of routes in some data structure would be the most efficient method to route packets along routes that had links constantly changing. This would allow for multiple packets to make usage of the same route

information that was only computed once instead of the same route information having been computed X number of times for X different messages. The best way would be to implement a data structure that held a reference to a node and the plans for this node. The plans for each node would give information on how to reach every other node possible on the route. This would result in something that would resemble a matrix. Routes would still be constantly updated in order to reflect the changing link weights and account for new shortest paths, however, this would still be more efficient than if the route was calculated by each node every time it received a new packet. The most difficult portion of implementing a general routing plan would be synchronization/thread safety. Ensuring that nodes would not access the routing plan necessary for an outgoing packet while the routing plan is in the process of updating could be tricky to implement, and also tricky to ensure that access is not needlessly restricted.

Q5) Utilizing a minimum spanning tree introduces both efficiency and inefficiency to our problem. A single minimum spanning tree could be kept track of for packet routing through a system, which would reduce the number of times a new route would have to be calculated. As the links changed, there would only be one run through of the MST algorithm in order to calculate this for the entire graph. This would reduce overall computation and more than likely lead to a speedier program. However, this dissemination scheme does come with trade offs. While using Dijkstra's would require the nodes to calculate different routes to every other node, Dijkstra's could result in a shorter overall path. MST focuses on connecting the entire graph and therefore sometimes the paths are not very direct and a packet would have to go through many nodes in order to get to its destination. Implementing a scheme that utilizes Dijkstra's could result in scenarios in which packets travel through a minimal number of nodes which could result in less data loss. The minimum spanning tree algorithm would be more appropriate if the nodes were to calculate the shortest paths instead of a registry since the nodes only are aware of their connections to other computers. Given the problem specifications in which the registry computes the shortest paths, I would continue to stick with an implementation of Dijkstra's.