Network routing in a faulty network

Paper by Nicholas Ward, Alan Garcia, and Kiara Compton

Introduction

In a network, packets of data are sent from a source node to a destination node while most likely needing to be passed from several intermediate nodes in the process. In order to optimize the throughput of how much data is sent to its destination, we should consider aspects like routing to the shortest path possible in the shortest amount of time possible. However, we must also consider another issue that could occur in a network. What if a node in a network fails and is unable to function? In this case the network should be updated in such a way that the failed node is avoided. However, what is the most effective way of updating the network that the throughput suffers the least? Our project tested two possible solutions for this problem.

Using Python, we simulated a network using a graph with weighted edges representing the time needed to send a packet from one end of the edge to the other. Nodes in this network may fail and the network will have to update itself accordingly to maximize the throughput.

Simulation design details

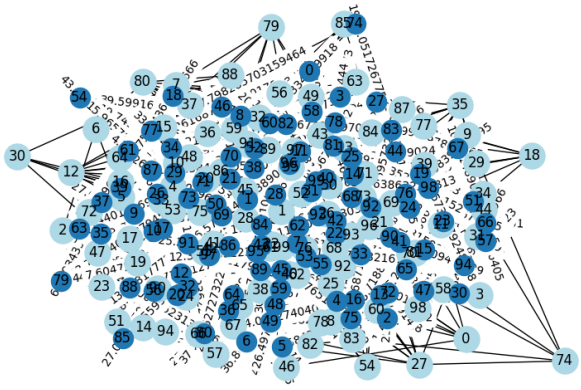

Figure 1: Graph

The first thing we did was create a simulated network. In our case it is simply just a graph with nodes, edges, and weights for the edges. Figure 1 is quite a mess but is an example of a randomly generated graph with 100 nodes labeled 1 to 100 and with weights between 1

and 50. The graph was created using a python library called NetworkX, extremely convenient for simulation of graphs. In order to simulate a failed node, each node was assigned a probability of failure. This failure probability was randomly chosen between 0 and 1. Then using another random number, we can randomly decide which node has failed or not, that failed node will be the one we try to avoid. Node failure handling will be discussed in the Node failure handling section below.

Node failure handling

When a node fails, not only should it be removed from the graph of the network, the network itself needs to update to take into account the node's failure. What needs to be updated is the routing paths that a node uses to send packets to a destination node. As our novel contribution to this project, we are not focusing on the question of what algorithm is the best at handling this problem, but rather what is the best reaction to a node failure. In our case we examine two possible reactions, every node in the network must rerun their routing algorithms without the failed node, or only the nodes adjacent to the failed node must rerun their routing algorithms.

To explain this in more detail, we'll discuss the routing algorithm that we've chosen. Dijkstra's algorithm is the routing algorithm that we've chosen for this experiment though the way we have implemented it is somewhat different. Rather than having the algorithm return the entire path that the packet must travel in the estimated shortest path, it will instead return the first node in the path, that is the adjacent node to the source that is the first node the packet is forwarded to in the shortest path. The idea is that if all the nodes have their own table of first nodes in the path to a destination node, then the shortest path could be taken implicitly by forwarding the packet to the first node in the path at each node. For example, if a shortest path is A-C-G-F then node A only knows that to reach F in the shortest path it must forward the packet to C. C, having received the packet, knows that to reach F in the shortest path it must forward the packet to G. G, having received the packet, knows to forward the packet to F as part of the shortest path.

The purpose of this functionality is primarily to allow for the second option to be possible, only the nodes adjacent to the failed node need to update their routing tables. This could also potentially save on memory. It is here where the potential problems with either method can be seen. If all nodes update their routing tables, this could potentially take up too much time as (in a more realistic application) the time it takes to receive new information about the network (i.e.

updated list of nodes and edge weights) and then run the routing algorithm may be too much even if each node performs these updates independently of one another and possibly parallel to each other. If only the nodes adjacent to the failed node update their routing tables, it could potentially be the faster problem however, there is no guarantee that the new path that is implicitly generated is the shortest path as the route only changes at the point where the failed node was encountered and does not take into account all of the potential paths from the initial source to the destination.

Results

There will be two strategies that we are going to be analyzing. Strategy 1 (S1), where every node in the network must rerun their routing algorithms without the failed node. Strategy 2 (S2), where only the nodes adjacent to the failed node must rerun its routing algorithms. To set up the experiment for Figure 2, a random source and destination node were chosen. Then we would calculate the time for the shortest path from the source node to the destination node found by Dijkstra's Algorithm. We can then compare the elapsed time for both S1 and S2, and decide which one would run faster.

Figure 2 showcases the packet times for both S1 and S2. On average S2 would outperform Strategy 1 where it would routinely have lower packet times from source to destination.

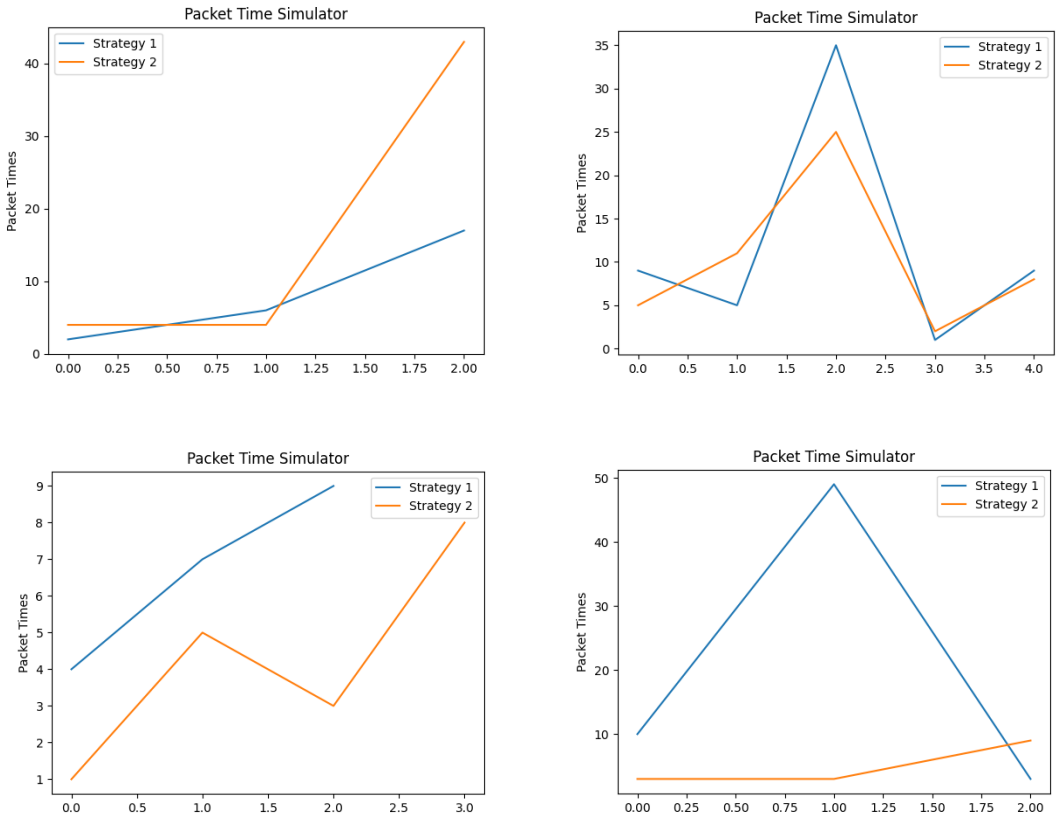

Figure 2: Four runs of example packet times on different graphs and failed nodes

Table 1 showcases the elapsed time to reach from source node to the destination node for both S1 and S2 with the different failed nodes. On average the total elapsed times for S2

tend to be shorter. Where we have a failed node does affect the elapsed times as every once in a while S1 will have a shorter time than S1.

Table 1: Ten runs of elapsed times for Strategy 1 (S1) and 2 (S2) and the failed node

Elapsed Time - S1 (s)	Elapsed Time - S2 (s)	Failed Node
0.204	0.177	0
0.434	0.638	32
0.494	0.339	35
0.533	0.257	53
0.433	0.266	6
0.283	0.126	15
0.263	0.198	50
0.393	0.496	27
0.352	0.586	70
0.472	0.285	65

Conclusion

In conclusion we created a network of nodes and edges with corresponding weights to simulate a network, we tested two strategies for node failure networking. S1, where every node in the network must rerun their routing algorithms without the failed node. S2, where only the nodes adjacent to the failed node must rerun its routing algorithms. It was found that on average the time from source node to destination node was faster in S2 than in S1. This coincides with our hypothesis that rerouting the entire network is more time consuming than that of rerouting the adjacent nodes of the failed node.