University of Nevada, Reno
Computer Science and Engineering Department

Final Project: Dynamic Routing Mechanism Design in Faulty Network using Simulated Route Request Messages and Replies

Contributors:
Mason Adams
Olliver Aikenhead
Davis DeSarle

Table of Contents

Protocol Functionality

Networks are everywhere in daily life, from phones, to cars, to personal homes, and even some appliances. With networks being such an important part of daily life, the connections to the networks must remain activated and accessible. It is inevitable that links between nodes in networks can fail, and a solution must be found to easily improve the recovery time of these connections. We used a recursive breadth-first search to find the neighbors of the nodes, to find a path to the destination. The program simulates a route request, and finds every neighboring node to connect to. It takes the first path it finds to the destination and then simulates replying to that information through the same path.

To simulate nodes failing, the edges of the graph, or “connections”, are destroyed at random, causing impossible routes. Edges are then added at random as well, simulating a network attempting to fix itself. These new connections are then traversed, finding the fastest path to the destination node. For example, if the path from A->J was ABCDEFGHIJ, and the connections between nodes DEF were removed, but a new path was created from C -> J, the new found path would be ABCJ. Our program proves the existence of these paths by attempting to get to the path through any means possible, then attempting a “reply” by going through the shortest path possible out of all of the paths. This determines the shortest path, and consistently finds new connections to get to the destination from the source.

This completely simulates the realistic scenario, when a link can become faulty at any time, but the source message must still be able to reach the destination. With our fix, finding the connection with the least amount of nodes, the destination is reached effectively.

Error Handling Scenarios

Errors are bound to happen in any networking scenario, especially when nodes consistently fail. Our program simulates random nodes failing, and fixes them by pathing around the failed nodes.

```
for (int i=0; i<(rand()%10); i++) {
  NodeChoice genNodes = randSelectNode(network);

  while (ptrForNode(genNodes.nodeA, mapNode)->connected(genNodes.nodeB)) {
    genNodes = randSelectNode(network);
  }
  std::cout << "Destroyed -> " << genNodes.nodeA << " and -> " << genNodes.nodeB << "'s edges."<< std::endl;
  destroyEdge(genNodes.nodeA, genNodes.nodeB, mapNode);
}
```

Fig. 1: Nodes are randomly being chosen to remove connections.

Figure 1 demonstrates how the program simulates damaged or unavailable nodes, by destroying the edges in the graph connecting them. This method allows the program to demonstrate its value every time it is run, since the random choice causes each run to be completely different. The error is handled through our send RREQ function, by finding the nearest connecting node to path to.

```
if(neighbor_node->nodeName == destination)
{
    //destination found
    std::cout     << "Node " << neighbor_node->nodeName << " received a RREQ from Node " << this_node->nodeName
                  << " to get to this node, we begin RREP ["
                  << (neighbor_node->RREQ_str + neighbor_node->nodeName) << "]" << std::endl;

    //use same path for RREP
    neighbor_node->sendRREP(destination, src, (neighbor_node->RREQ_str).size(),neighbor_node->RREQ_str, destination);
}
else
{
    //did not find, so neighbor_node will now ask its neighbors
    std::cout     << "Node " << neighbor_node->nodeName << " received a RREQ from Node " << this_node->nodeName
                  << " to get to Node " << destination << ", list of letters: " << neighbor_node->RREQ_str << std::endl;
}
```

Fig 2: Error Handling by finding a path to the node through a neighboring node, or going through the neighbors' nodes recursively to find a new path

Figure 2 demonstrates how the new path is found to the destination nodes, since the destination path can change with destroyed edges. It then uses send RREP to repath back, simulating a reply, verifying that the path functions as intended. If no path is found to the node, then the program ends, but with the constant new creation of edges and connections, there is a high chance of catching any error in the system.

Novel Protocol and Visual Simulations

A novel contribution that we included in our project was randomization of node selection. In our code we are randomizing the time for our simulation and randomly selecting one of the generated nodes to connect an edge to. The two connecting nodes being selected are then randomized by dividing rand with the remainder of the number of nodes in the network. Node edges are destroyed by being randomly selected in the same manner, this will simulate a fault somewhere in the network. Figure 3 shows the mapping of nodes while edges are discovered in the fixed network simulation.

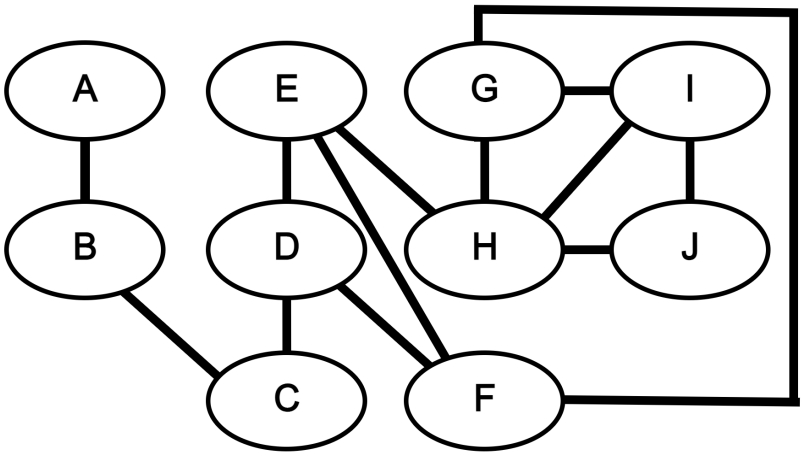

Fig. 3: Demonstration of the Fixed Network Simulation

Figure 3 is a demonstration of the beginning of our program shown in figure 5 and stated as the network fixed simulation. The purpose of this figure is to show the links between nodes and how our program simulates route requests (RREQ). As previously described in protocol functionality our program attempts to find the fastest route to the destination node. The nodes are equally weighted and RREQs are sent by alphabetical order of what comes first. So, node B receives a RREQ from node A, node C receives one from node B and so on.

Figure 4 demonstrates the different types of connections between nodes as well as the RREP traceback. The green connection between nodes is an accepted RREP from the receiving node. A red connection is an ignored RREQ due to the receiving node forwarding the RREQ already. A black connection is a RREQ that is not sent due to it already being at the destination node. The highlighted blue nodes are the fastest path from node A to node J. It also demonstrates the traceback path the RREP takes.

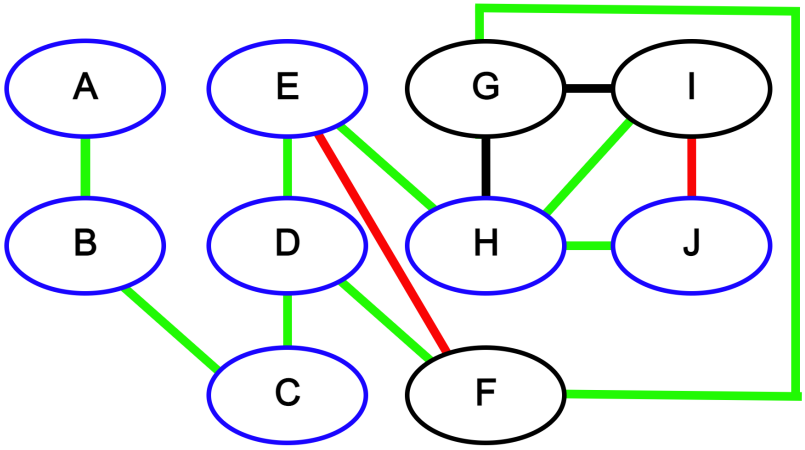

Fig. 4: Shows the path the program follows to find the fastest path to J

Results / Analysis

Running the program through the make file demonstrates the effectiveness of the algorithm to pathfind to the destination node, even with deleted nodes. The first output prints what happens in an ideal scenario, where all of the edges in the graph are connected and the path can easily be found.

```
-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
Network Fixed Simulation:

Route Discovery from node A to node J
Node B received a RREQ from Node A to get to Node J, list of letters: A
Node C received a RREQ from Node B to get to Node J, list of letters: AB
Node D received a RREQ from Node C to get to Node J, list of letters: ABC
Node E received a RREQ from Node D to get to Node J, list of letters: ABCD
Node F received a RREQ from Node D to get to Node J, list of letters: ABCD
Node H received a RREQ from Node E to get to Node J, list of letters: ABCDE
Node G received a RREQ from Node F to get to Node J, list of letters: ABCDF
Node I received a RREQ from Node H to get to Node J, list of letters: ABCDEH
Node J received a RREQ from Node H to get to this node, we begin RREP [ABCDEHJ]
Node H received RREP from J to A with route [ABCDEHJ]
Node E received RREP from H to Node A with rout [ABCDEHJ]
Node D received RREP from E to Node A with rout [ABCDEHJ]
Node C received RREP from D to Node A with rout [ABCDEHJ]
Node B received RREP from C to Node A with rout [ABCDEHJ]
Node A successfully received path to J route [ABCDEHJ]
```

Fig. 5: Demonstration of the initial network simulation, where the network is in full functioning order.

Figure 5 shows how our recursive RREQ function finds the nodes it needs to traverse from the source in the fixed network simulation. This example shows the program finding every connection from A to J, and the preconceived network shown in figure 4 demonstrates how the program gets to that path. Once it successfully finds the path, it simulates an RREP (reply) and goes back down that same path.

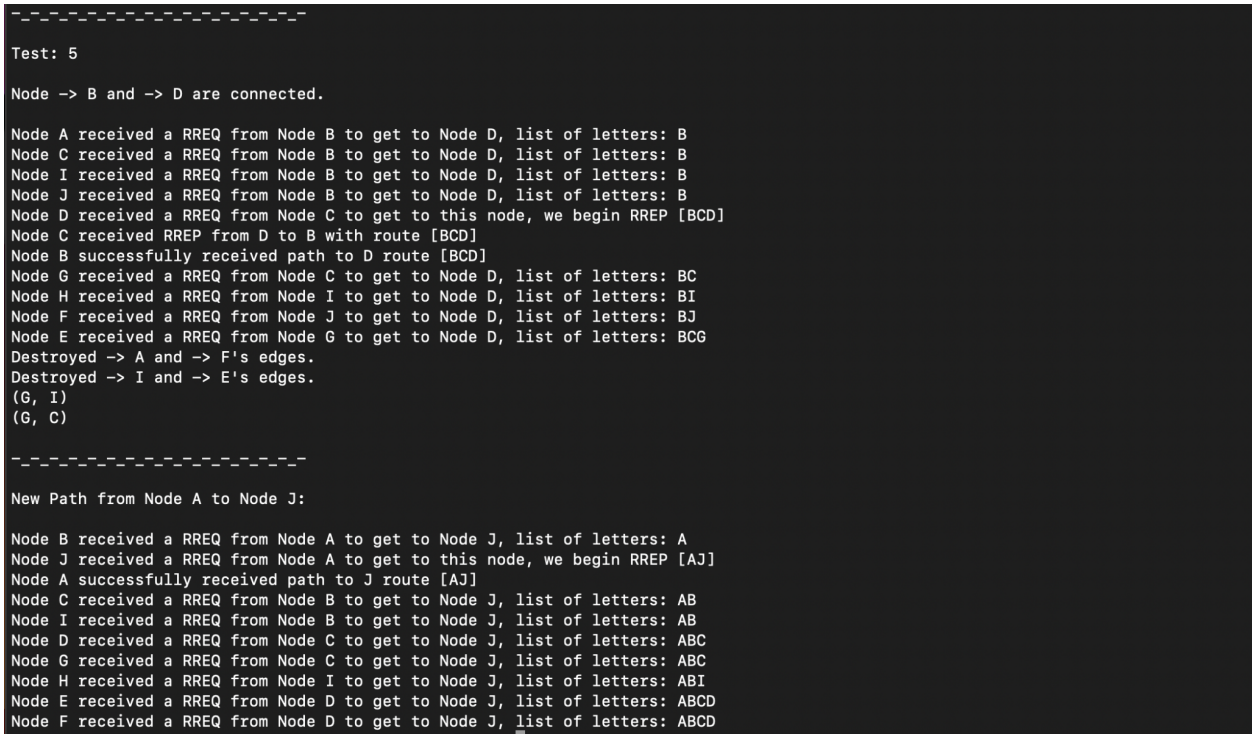

Fig. 6: Demonstration of the initial network simulation, where the network is in full functioning order.

Figure 6 shows the end result of the project, showing the program finding the fastest path with five rounds of damaged and repaired links. The program is able to path from A to J through the deleted nodes, finding a path directly from A->J. This path was created in figure 7, found through the RREQ response, which attempts to pathfind using the least amount of nodes.

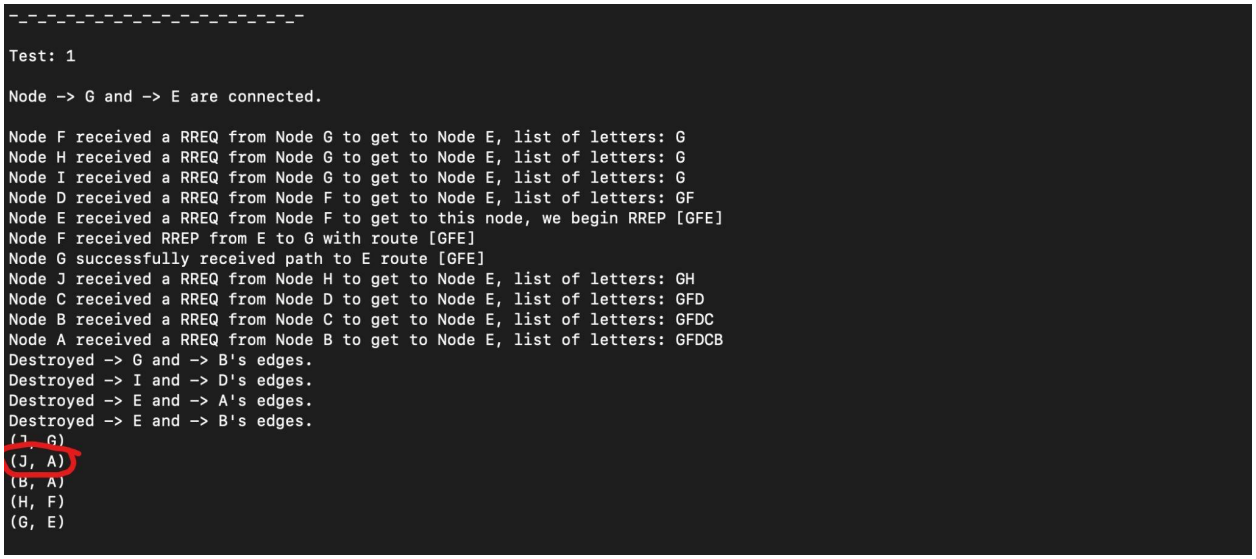

Fig. 7: Shows the path created to get from A->J directly