Dynamic Routing Mechanism Design in Faulty Network

CPE400

Sybille Horcholle, Nicholas Mason, Nicholas Thom

CPE 400

Sybille Horcholle, Nicholas Mason, Nicholas Thom

December 3, 2018

Dynamic Routing Mechanism Design in Faulty Networks

**Introduction**

The topic addressed in this project finds a creative solution for dynamic routing in faulty, mesh networks.  In any given network, whether it is a personal area network (PAN), wide area network (WAN), or local area network (LAN), nodes are connected by links to create a mesh network.  Different nodes in a mesh network can communicate with each other by establishing links between nodes. By communicating with a neighbor node, a particular node can find the most efficient path to get data to and from a desired node.  In many cases, nodes and links fail intermittently. When simulating a mesh network, it is important to implement early detection of link and node failure in order to avoid a networking failure. This code will label “dead-end” nodes to improve the efficiency of finding the quickest path from one node to another.  Dijkstra’s Algorithm will be utilized to ensure efficient and reliable results in finding the shortest path between nodes. In addition, the code will take the probability of a node failing into consideration when applying Dijkstra’s Algorithm to find the shortest path possible. If a node is prone to failing, that particular path will not be utilized.

**Functionality of the Protocol**

This code has been written in the Python scripting language to ensure easy readability of the source code and show the functionality of the code by implementing a simulation of the mesh network.  Dynamic Source Routing (DSR) is the primary platform used in the creation of this project. The code implemented in this project takes a DSR platform, and incorporates Dijkstra’s Algorithm to find the shortest path possible between a start node and an end node.  A critical concept to handle in this project is failure between nodes in the shortest path between a start and end node. Each node is given a probability to fail at the beginning of the simulation, which will be stored for the program to consider when creating the shortest path.  The node ID, its neighbors, and the percent change of failure will be read in from a text file at the start of execution. A second number is randomly generated for each node, resulting in a node failing if it is less than or equal to the first number given in the text file.

In addition, a failure flag has been initialized to let the program know immediately upon starting which nodes have failed.  An active flag allows the program to know which nodes are safe to use in the process of transporting data. Additional code functionality for the node class includes checking to see if a given node has a neighbor, and what information the node holds.  Checking to see if a neighbor has been visited has also been implemented to ensure a node can visit all its neighbors efficiently. This gives the program a better chance of avoiding the faulty nodes entirely, choosing a more efficient and safe path to transport data.

The program starts by reading in the ID, the neighbors, and the percent change of failure for each node given in a text file. The programmed algorithm starts by prompting the user for a start and end node. From there, the program will compare the percent chance of failure between the starting node’s neighbors, choosing the neighbor with the smallest percent chance of failure. In addition, there will be a check to see if the desired end node is one of the current node’s neighbors. If that is the case, that node will be chosen, regardless of its percent chance of failure. There will also be a flag that will identify if a neighboring node is a dead end node, which means the program will not consider that an option for travel unless it is the desired end node. Once a node is chosen, a random number will be generated, and if that number is less than the designated percent chance of failure, that means that node has failed. If it fails, the program will look at the current node’s neighbors again, ignoring the node that has just failed. The algorithm will continue, by checking for the desired end node from the current node’s neighbors, choosing the node with the smallest percent chance of failure, and seeing if that node fails, until the end node is reached.

Additional functions have been added to ensure that the program finds the shortest path possible from a start node to end node. One function has the ability to find the minimum distance between two nodes. This will be utilized in the improved shortest path algorithm in determining which path from the starting node to the end node will be most efficient. In addition, the shortest path algorithm has the functionality to check all the nodes to see if they are active or if they have failed. When creating the path from start to end, if a failed node has been encountered, the algorithm will ignore that node and find a new route. All of these factors will be taken into consideration when determining the most effective path between nodes.

**Novel Contribution**

When a node fails, the first principle a network uses to resolve the problem is to find a new route through a different link to the needed node. The novel contribution added to this program is the ability to find a faulty node that may appear in the path from start node to end node. To do this, the program considers the percent chance of failure of a given node, along with creating a dead end flag assigned to a node with only one neighbor. Creating a flag for dead-end nodes will allow the program to only venture to that given node on a given path if it is indeed the start node or the end node.  A dead-end node has been defined in the program as a node that only has one neighbor node. The implementation of Dijkstra's Algorithm will allow the simulator to choose the best routing path to achieve the optimal efficiency. Incorporating various flags for faulty nodes and links, along with dead-ends, will allow the program to find the shortest path between the start node and end node in a given data packet transfer, without having to worry about faulty nodes in the path.

**Results and Analysis of Results**

Through the process of simulating a mesh network, along with incorporating the functionality presented in this project, faulty nodes and links were able to be detected and avoided in the data transfer process. As a group, we decided to test our skills using Python, which we had little experience in to begin this project. After spending time setting up the node class and working on the shortest path algorithm, the syntax of the programming language became more familiar. In addition, the use of the matplot library allowed us to give visuals to show the success of our project.

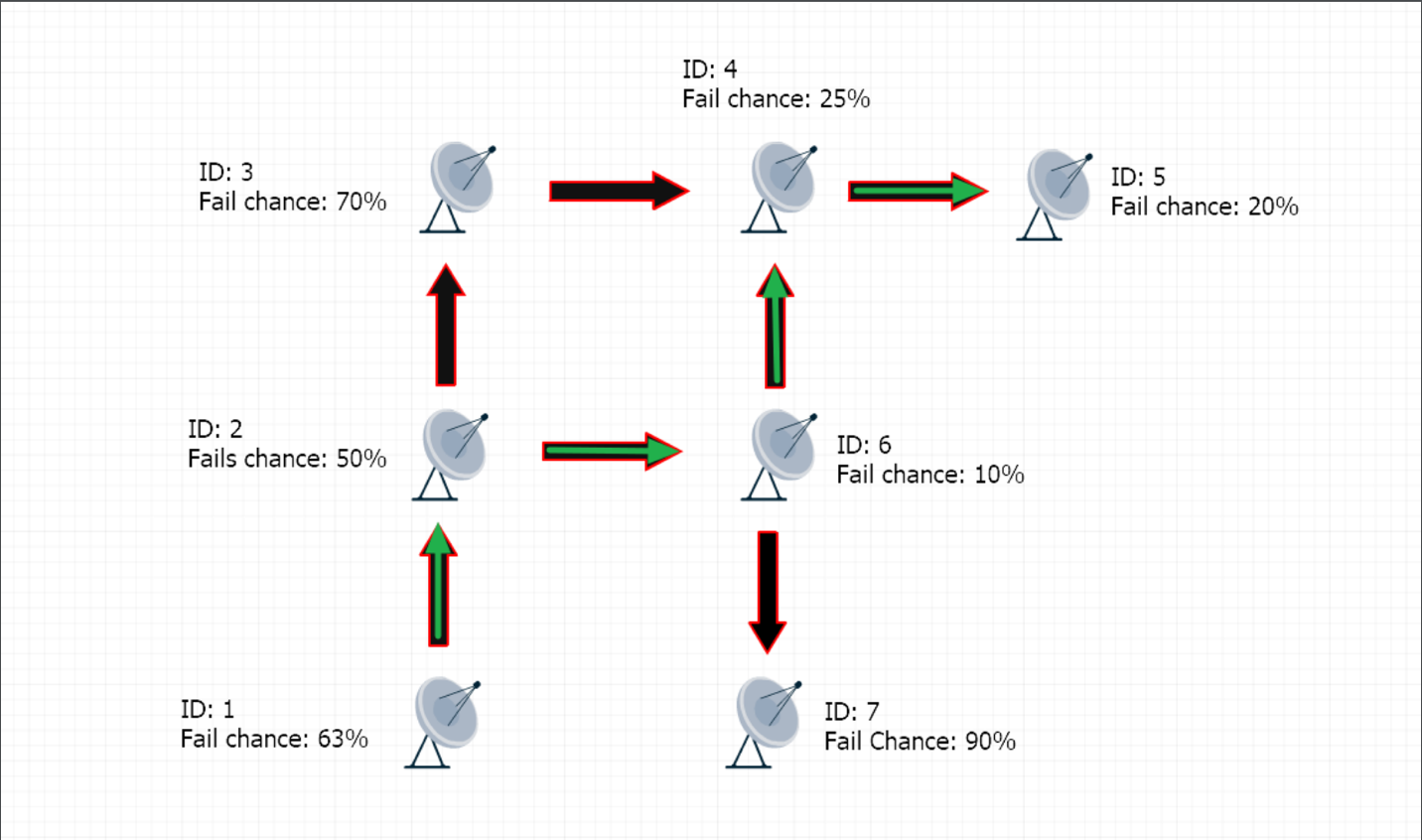


Fig. 1: Figure 1 shows the results of being able to detect the probability of failure of a given node, resulting in the shortest path possible.

Figure 1 depicts the simulated mesh network using the matplot library built into the python programming language. As seen above, the user has requested to send data from node ID 1 to node ID 5. The packet starts by moving from ID 1 to ID 2, where it now has a choice to which node it will visit next. The modified shortest path algorithm allows the program to check the percent chance of failure of node ID 3 and node ID 6, each being neighbors of ID 2. The program chooses to travel through ID 6 because it has a smaller percent chance of failure at 10%, rather than ID 3 that has a 70% chance of failure. From there, the packet knows that node ID 7 is a dead end due to the dead end flag added to the functionality of the code, resulting in data traveling through node ID 4. Finally, node ID 4 is able to transfer the packet to its end node of ID 5. Checking the lowest percent chance of failure allows for the shortest path possible.

Fig. 2: Figure 2 shows how each node obtains a given ID, neighbors, percent change of failure, and the path it takes from the improved shortest path algorithm

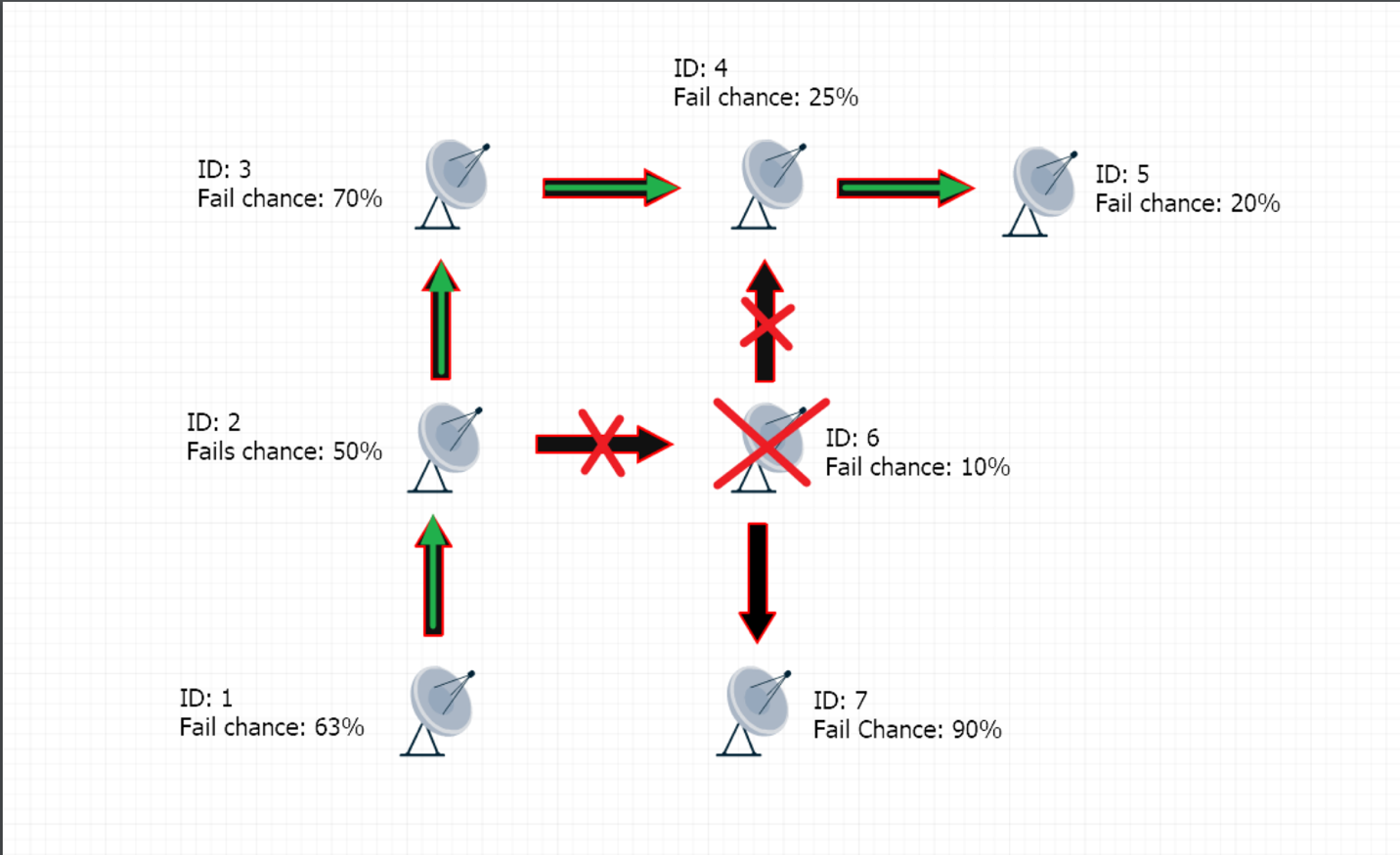


Fig. 3: Figure 3 shows a circumstance where node ID 6 has failed, resulting in the program choosing node ID 3 for the shortest path from ID 1 to ID 5.

Figure 3 shows an example where a node has failed when looking for the shortest path from start node to end node. Similar to the example in fig. 1, the goal is to get from node ID 1 to node ID 5. Like in fig. 1, node ID 1 leads to node ID 2. Now, however, node ID 2’s only option is to go the node ID 3 because node ID 6 has failed, as seen in fig. 1 above. After that, the program proceeds in the same way as before, leading all the way to the end node, which in this case was node ID 5.

**Conclusion**

After simulating various test cases with a mesh network, the error detection programmed into the code allowed the program to detect faulty nodes when reading in the information on each node from the text file at the beginning of execution.  Upon realizing that a faulty node exists on a given path from start node to end node, the program chooses to find another path, resulting in the shortest path possible because the path will avoid dealing with failed nodes completely. When creating a route from start node to end node, it is always more important to ensure that the information is transmitted safely if that information is important. In some cases, if the shortest path is always necessary, this improved shortest path algorithm will ensure that the packet is transmitted at the highest rate possible, on the shortest path. The code applies Dijkstra’s Algorithm, in addition to looking at the probability of failure, in order to find the shortest path.