Dynamic Routing Mechanism in a Faulty Network

Crystal Atoz, Korben DiArchangel, Joshua Insorio
CSE Undergraduates, University of Nevada, Reno
CPE 400 Project

Abstract

In a mesh network, nodes have to relay their communication through other nodes in order to communicate. During the communication process, nodes and links may fail. This report discusses a possible solution to failing nodes and links by simulating a mesh network that will reroute to avoid faulty nodes and links. The description and algorithms section discusses the algorithms and code used in depth. The data and results section analyzes the solution with example tests.

Introduction

The intent of this project is to find a way to make mesh networks more efficient if nodes or links were to fail. If two nodes were communicating with each other and a node or link on their path were to shut down, a rerouting algorithm would have to be used in order for those nodes to communicate with each other again; this is done by updating the forwarding tables in the nodes. A solution that guarantees the shortest path for all nodes would be Dijkstra's algorithm, but this particular algorithm takes a long time to process, meaning that two nodes that have a possible alternative route will be left disconnected for hours in a larger network. A new routing algorithm, which involves the significantly faster Breadth First Search, is used to calculate a less optimal, but viable path for the nodes to connect with each other. As BFS does significantly less operations and is only applied to connections that need to be rerouted, this algorithm is expected to fix routes faster while Dijkstra's algorithm is making computations. It is also expected to run faster as Breadth First Search has a runtime of O(V+E), while Dijkstra's algorithm has a runtime of O(V+E log(V)); this will matter significantly in larger networks with thousands of nodes. [2]

Description and Algorithms

When the simulation is executed from the command line, the user is prompted to enter the provided text file. A list of commands are then displayed: seenodes, seelinks, flipnode, fliplink, fixnode, fixlink, findroute, and stop. These commands are shown in Figure 1.

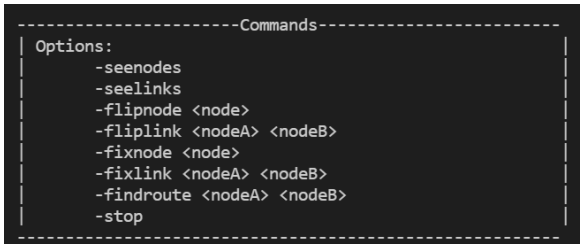

Figure 1: The above image displays the commands of the program.

Seenodes displays all nodes that were created. Seelinks displays all links that were created. Flipnode/fliplink shuts down a node/link if it is active and will activate the routing algorithms. Fixnode/fixlink will repair a broken node/link, but this does not activate the routing algorithms. Findroute finds a path between the chosen source and destination node using the node's forwarding table. Stop shuts down the program. These commands are all built to handle possible input errors; it will refuse to activate the command if the nodes/links submitted do not exist, and it will reject commands that aren't in the ones listed. In addition, commands such as flipnode and fixnode cannot be used while Dijkstra's is running to assure no errors occur, though the user can still use seenodes, seelinks, and findroute, as those are read-only functions.

The nodes and links in the text file are then created as shown in Figure 2. The user can then use the commands provided to display, shut down, and find routes between the nodes and links they have created. Each node will have a list of neighbors as well as a forwarding table that allows the node to access any other node on the graph, if possible. The user can see and find routes between the nodes, but they cannot add or remove nodes/links during runtime; this was done in order to prevent conflicts with the forwarding tables and path algorithms.

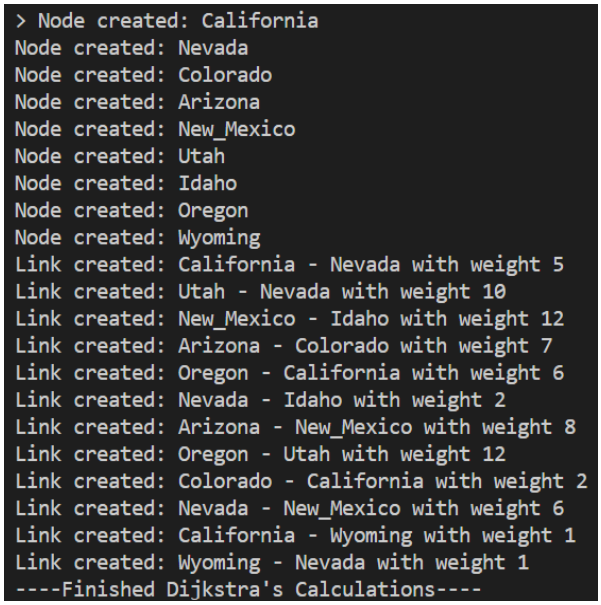

Figure 2: The above image shows the creation of all the nodes and links in the input file.

The nodes and links can fail in two ways; the user can either use the input commands or the program will automatically shut down nodes/links periodically. If a node or link were to fail, the BFS algorithm will be used to calculate an alternate route that is faster than recalculating Dijkstra's algorithm. Figure 3 shows the program automatically shutting down/fixing nodes and running both functions. The BFS algorithm is used on nodes/links affected by the shutdown to determine a quick path, though it may not be the most efficient. In the case of the simulation, only a few nodes are used which does not properly illustrate the time complexity of rerunning Dijkstra's algorithm, implementation of a small "lag" was added to simulate this.

```
Node Idaho has been shut down.
Link Nevada Idaho has been shut down.
----Finished Fast Route Calculations----
----Finished Dijkstra's Calculations----
Node Arizona has been shut down.
Node California has been shut down.
Node Idaho has been repaired.
Link Nevada Idaho has been repaired.
----Finished Fast Route Calculations----
----Finished Dijkstra's Calculations----
Node Arizona has been repaired.
Node California has been repaired.
Node Utah has been shut down.
Link California Nevada has been shut down.
Link Oregon Utah has been shut down.
Link Colorado California has been shut down.
```

Figure 3: The above image shows nodes and links failing and repairing themselves.

Dijkstra's algorithm finds the optimal path between the source and destination node by finding the minimal weighted path. A graph is created with all listed nodes and links that are active is initialized. The algorithm then traverses the nodes and updates the distance values of the adjacent nodes. The lowest weighted path is noted and updated during this process, which is aided by a function that finds the node with the minimum distance value. After completion, the optimal path between the requested nodes are printed. A forwarding table is then created after Dijkstra's algorithm is calculated the first time. BFS works similarly, though it disregards the length of the links and instead tries to find a path that works. It will update the forwarding tables if a path is found, and if a path isn't found, it will modify the forwarding table to tell the node that the end node is impossible to reach. Unlike Dijkstra's algorithm, BFS is only applied to paths that visit shut down nodes/links, which saves a significant amount of time, as most paths will likely not use the node/link.

Both algorithms will update the forwarding table by adjusting the start node's forwarding table to point to the second node on the path. The algorithm guarantees that the nodes will go towards the goal; Dijkstra's algorithm always calculates the most optimal route and will output the same path every time, and BFS will always output the path that takes the least amount of nodes to travel to; it will never go backwards. Figure 4 shows how the algorithms update the forwarding tables.

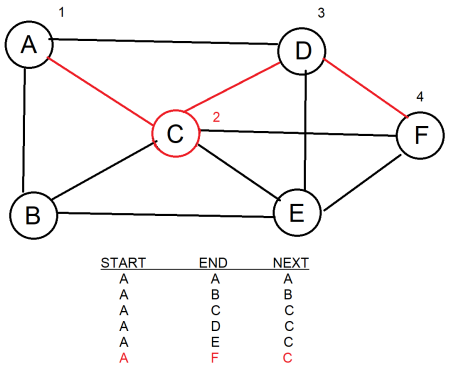

Figure 4: A diagram showing how the algorithms calculate the path, and then use the second node in that path to update the forwarding table.

Threads are used so that Dijkstra's algorithm and BFS can run at the same time. Mutex locks are used to assure that neither of these algorithms interfere with each other; the forwarding table is only updated within the mutex lock. Dijkstra's algorithm is run outside of the mutex lock, but it stores all of its results in a temporary forwarding table so that it doesn't interrupt the BFS algorithm. The temporary table is then applied to the real forwarding tables inside the mutex. Dijkstra's algorithm also never reads from

the forwarding tables so that BFS is allowed to do anything with them.

Data and Results

A few examples are conducted in the mesh network and discussed. The data and results of the solution will also be detailed.

After the nodes and links are created from the test input file, a request was made from California to Arizona. Dijkstra's algorithm calculates the optimal path. By manually turning off the Colorado node, with the flipnode command, Dijkstra's algorithm is then recalculated and the forwarding table is updated. Another request to find a route from California to Arizona is made and the optimal path is displayed as shown in Figure 5.

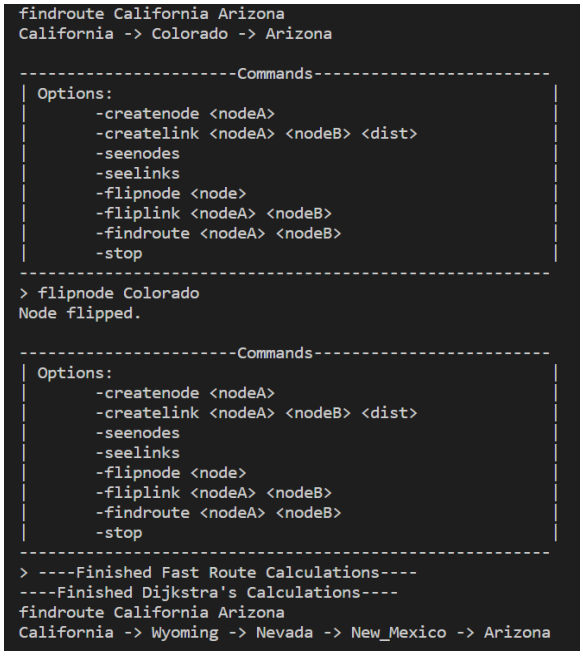

Figure 5: The above image shows the optimal path between California and Arizona when Colorado is on and off.

Figure 6 shows an example of our node link shutdown and recalculation is presented. Here the link between Nevada and Idaho has been shut down manually via the flipnode command. Immediately the BFS algorithm follows up and is calculated to supply a new route. The novelty of the solution itself is in the ability to guarantee an alternate route at a much shorter memory cost.

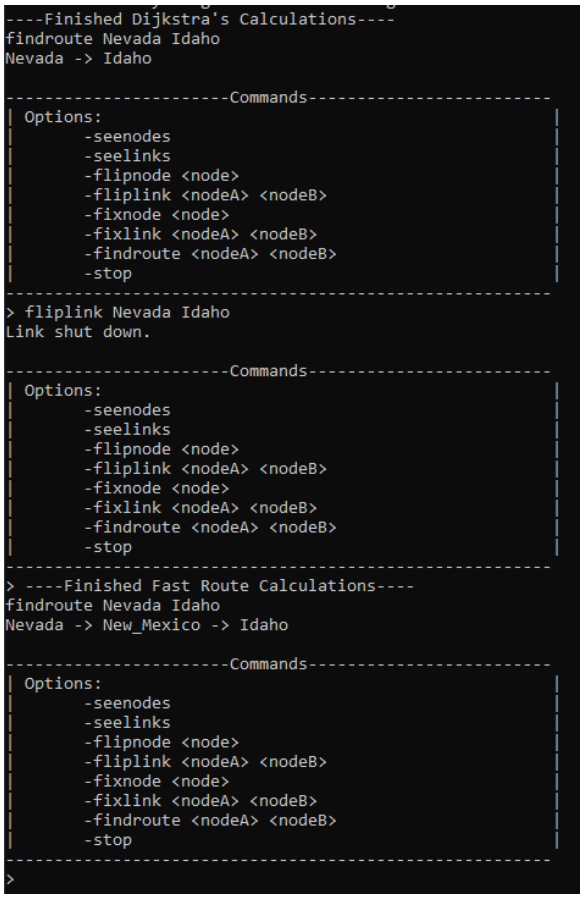

Figure 6: The above image shows a nodelink shutdown and recalculation.

With the previous mentioned examples, along with other tests, it was found that the algorithms used are very efficient. During production of the solution, over a hundred routes were tested and the BFS algorithm would always result in an efficient or

alternate route being formed if possible, and Dijkstra's would consistently produce an optimal path. Furthermore, the program will operate on its own and is capable of consistently producing forwarding tables without issue, as it was tested over a 45 minute period without any errors.

Conclusion

In the data gathered and through the multiple stress tests, it can be concluded that the solution is effective. A strong emphasis on the ability to provide an alternative route provides a unique way of keeping nodes connected when a node or link is broken. The project has increased the team's understanding of mesh networks and how to successfully move data in a faulty network through alternate routes. In the case of a larger mesh network simulation, this solution can be used to continue to test its efficiency and reliability. Although mesh networks can be unreliable with how dynamic they are, it was found that it is possible to increase the speed and reliability of the network while lowering memory cost through our solution.

References

[1] J. Kurose, K. Ross, "Computer Networking: A Top Down Approach," *Pearson*, Seventh Edition, 2017.

[2] Sryheni, Said. "Difference Between BFS and Dijkstra's Algorithms," *Baeldung*, 2020. https://www.baeldung.com/cs/graph-algorithms-bfs-dijkstra