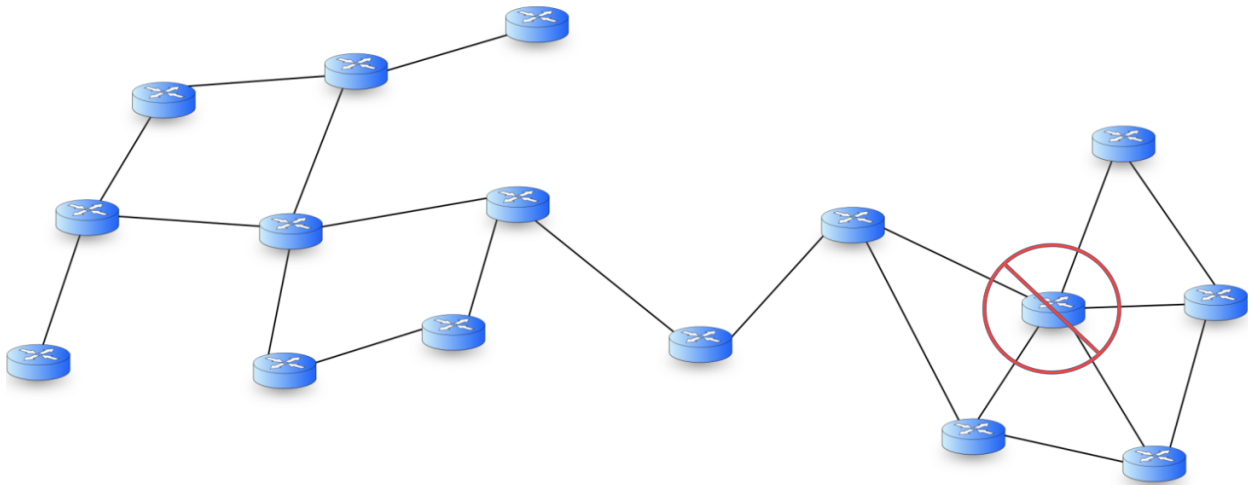


(6) Dynamic Routing: Faulty Network



Abraham Meza, Harrison Hong, Alexander Mathew

Date: 05.03.2021

Course: CPE 400 - 1001

Instructor: Engin Arslan

University of Nevada, Reno

Table of Contents

Table of Contents	1
Program Setup and Usage	2
Introduction	2
Protocol Functionality	3
Conceptual Analysis of Routing Protocols via Mesh Networks	3
Implementation of Routing Protocols via Mesh Networks	3
Out of the Box Thinking: A Novel Contribution	14
Results and Analysis	14
Conclusion	15
References	15

Program Setup and Usage

In this project, pip3, a version of pip installer for Python 3 is utilized. In order to utilize pip3, Python 3 must be installed first. This can be done using the provided download file found on the Python website. After this, it is necessary to install the pip3 installer from the pip documentation website. Once finished, the “pip3 install” command can be used to install the following libraries: combinations, groupby, string, random, sys, time, math, queue

The libraries needed that do not require pip3 are the following:

networkx, matplotlib.pyplot, matplotlib.animation

That is the required setup needed prior to running the project. Once the file “main.py” is downloaded, simply run the command “python3 main.py”. This will run the project.

Introduction

As a networking tool, mesh networks are highly interconnected networks composed of nodes and devices. As a local network topology, the infrastructure nodes connect directly, dynamically, and non-hierarchically to as many nodes as possible as a means to communicate data and information between clients. However, in comparison to other network systems, mesh networks promote flexibility and scalability by implementing self-healing algorithms in which the network dynamically excludes faulty nodes, identifying new efficient routes in the network using shortest path algorithms to preserve energy and data transmissions [Mesh Networks, website]. As a self-organizing network, mesh networks are appropriate tools for stable communication as it promotes convenience through fast recovery and response times.

In this project, the team will simulate the dynamic routing capabilities of mesh networks where there exists intermittent failures of nodes and links due to energy consumption (i.e., power outages, temperature, hardware failure, etc.) and network congestion. In response to the failure of systems, the network will adapt and reroute data transmissions to reflect optimal performance while indicating the current state of the network to monitoring users.

The following sections will illustrate the group’s protocol functionality, novel contribution, and analyzed results — summarizing their learning objectives and observations.

Protocol Functionality

As an important network paradigm, mesh networks offer a cost-effective method of internet connectivity through their dynamic approach as a self-organizing network (SON). In order to formulate a method of approach, the group observed a variety of optimal algorithms as they analyzed different routing protocol structures to implement a realistic network simulation.

Conceptual Analysis of Routing Protocols via Mesh Networks

For the project, the team utilized the Open Shortest Path First (OSPF) routing protocol, an interior gateway protocol (IGP), to simulate network routing communication between a set of routers in a single Autonomous System (AS) [OSPF, website]. Using Dijkstra's Shortest Path Algorithm to determine the most cost-efficient path between each router in the network, the group created a subnet map illustrating the connections of each root node to other routers in the system, creating a simulated environment tied to a predetermined source node. As a traffic-like structure, OSPF was the ideal routing protocol as its knowledge of topology allowed routers to calculate the most optimal path that followed a predetermined criteria, promoting a network's speed, stability, and scalability inside a singular domain [OSPF, website].

In order to simulate an OSPF routing protocol via a mesh network, the program creates a bidirectional weighted graph as a means to simulate a local area network (LAN) where each node is a separate router with weighted connections linking/mapping them together. Our simulated graph is made up of 26 network nodes labeled A-Z (ASCII Uppercase Symbols: recommended), where node A represents the source node and nodes B-Z represent destination nodes: all having a probability of failure due to energy consumption and/or network congestion. Once the program has created the initial network graph, the system will output Dijkstra's Algorithm for all nodes reachable through the source node, illustrating the most optimal path/distance as one traverses through the mesh network.

Implementation of Routing Protocols via Mesh Networks

Based on the group's program implementation, each network scenario is generated randomly based on the following predetermined conditions: probability, source node identification, the total number of nodes in the network, and the total number of faulty instances. Based on user configuration, the network will implement the appropriate attributes to generate a random undirected graph (Erdős-Rényi), and will then assign at least one random edge for each node as

means to create a connected network. For the purpose of documentation, the group's scenario utilizes 26 nodes labeled using the ASCII uppercase dictionary (A-Z), where node A is the source node and nodes B-Z are directly and indirectly connected to the source node before potential network failure.

Based on the total number of faults, the network will simulate a power outage in which node battery life is heavily diminished, causing node and link failures at an inconsistent rate. During each fault instance, there will be a random selection of nodes and links, where no more than a fourth of each attribute will start to fail. When presented with faulty networks, the mesh network will dynamically respond to the outage by removing faulted network connections and rerouting the remaining paths from the source node to the remaining active networks for the most optimal solution. During each instance, the program will display the most cost-efficient path of each available node to the user's graphical interface (GUI), while also informing the user of unreachable and disconnected networks from the source node. It is important to note that if the source node should fail due to insufficient battery power, the entire network will shut down until it is restored: printing an error message, apologizing to the user for the inconvenience.

As a means to dynamically optimize the network during faulted instances, the program utilizes a shortest path algorithm to traverse and identify the most optimal path from the single source node to all other connected nodes. For the purpose of the project, the group utilized Dijkstra's Algorithm to calculate the most optimal path for each instance as it has a low, and almost linear, complexity, calculating the most cost-efficient path for all positive weights in a multi-path environment. However, it is important to note that the Bellman-Ford Algorithm may be a better alternative for future project improvement as it calculates the shortest path for both positive and negative edge weights for dense graphical environments. Even so, since all weights in the network have some form of cost (i.e., battery consumption, transmission time, etc.), all weights will be a positive integer — validating the use of Dijkstra's algorithm in multi-path routing. Through the use of Dijkstra's algorithm, the network is able to actively identify the optimal path/distance of the source node to all other active nodes, while simultaneously avoiding faulty networks.

As a result of the network's complexity, the team implemented a GUI to display relevant and useful information through visual aids as the user refers to terminal prompts for observation and analysis. As seen in Fig. 1, the program creates a weighted, bidirectional mesh network consisting of 26 nodes labeled A-Z, where node A, the network's source node, is highlighted in yellow.

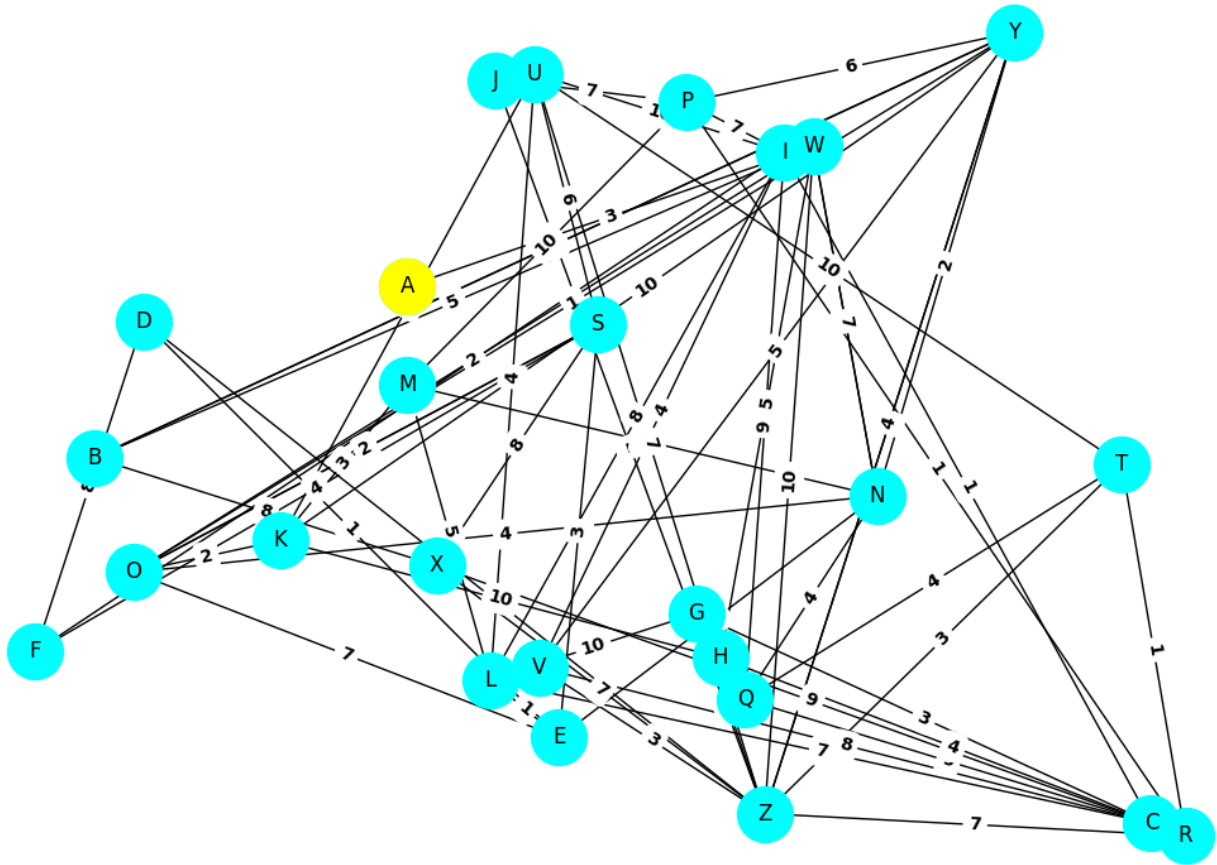


Fig. 1: A bidirectional weighted network randomly generated by the program's pre-determined network characteristics. Node A (highlighted in yellow) represents the source node, while nodes B-Z represent interconnected routers that belong to the same local area network (LAN).

Upon generating the network graph, the program will output the graph's statistical information, along with the most optimal path/distance for each connected node (Dijkstra's Algorithm) to the terminal to be used as navigational reference. Figure 2 illustrates the terminal prompts provided to the user for the generated graph.

```

////////////////////
//// Original Network ////
////////////////////

The mesh network source node is [A].

////////////////////
//// Network Information ////
////////////////////

Number of nodes: 26

Online Network Nodes: [A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z]

Number of edges: 67

Current Edge Pairings: [(A, W, {weight: 3}), (B, Y, {weight: 8}), (B, Y, {weight: 9}), (B, W, {weight: 5}), (B, X, {weight: 8}), (C, Q, {weight: 9}), (C, G, {weight: 3}), (C, E, {weight: 1}), (D, F, {weight: 8}), (E, N, {weight: 8}), (E, L, {weight: 1}), (E, O, {weight: 7}), (E, S, {weight: 3}), (F, S, {weight: 4}), (F, I, {weight: 9}), (G, Z, {weight: 5}), (I, W, {weight: 4}), (I, L, {weight: 8}), (I, P, {weight: 7}), (I, Q, {weight: 9}), (I, U, {weight: 10}), (I, V, {weight: 4}), (J, P, {weight: 7}), (J, U, {weight: 3}), (K, Y, {weight: 10}), (L, U, {weight: 4}), (L, M, {weight: 5}), (M, N, {weight: 4}), (M, P, {weight: 10}), (N, W, {weight: 5}), (N, W, {weight: 7}), (N, O, {weight: 4}), (N, O, W, {weight: 2}), (O, Y, {weight: 1}), (P, Y, {weight: 6}), (P, R, {weight: 1}), (Q, T, {weight: 4}), (R, Z, {weight: 7}), (R, T, {weight: 1}), (R, X, {weight: 9}), (S, U, Z, {weight: 7}), (V, Z, {weight: 3}), (V, Y, {weight: 5}), (W, Z, {weight: 10}), (X, Z, {weight: 7}), (Y, Z, {weight: 7}), (Y, Z, {weight: 4})]

////////////////////
//// Dijkstra's Algorithm w/ Online Networks ////
////////////////////

A to A (path cost = 0): [A]
A to B (path cost = 8): [A, W, B]
A to C (path cost = 8): [A, W, I, C]
A to D (path cost = 11): [A, W, O, S, E, D]
A to E (path cost = 10): [A, W, O, S, E]
A to F (path cost = 11): [A, W, O, S, F]
A to G (path cost = 11): [A, W, I, C, G]
A to H (path cost = 8): [A, W, H]
A to I (path cost = 7): [A, W, I]
A to J (path cost = 12): [A, W, O, Y, Z, J]
A to K (path cost = 7): [A, W, O, K]
A to L (path cost = 11): [A, W, O, S, E, L]
A to M (path cost = 10): [A, W, O, K, M]
A to N (path cost = 8): [A, W, N]
A to O (path cost = 5): [A, W, O]
A to P (path cost = 12): [A, W, O, Y, P]
A to Q (path cost = 12): [A, W, N, Q]
A to R (path cost = 12): [A, W, H, R]
A to S (path cost = 7): [A, W, O, S]
A to T (path cost = 13): [A, W, O, Y, Z, T]
A to U (path cost = 13): [A, W, O, S, U]
A to V (path cost = 11): [A, W, O, Y, V]
A to W (path cost = 3): [A, W]
A to X (path cost = 15): [A, W, O, S, X]
A to Y (path cost = 6): [A, W, O, Y]
A to Z (path cost = 10): [A, W, O, Y, Z]

* Creating network interface *
* Network interface has been downloaded for your convenience *

```

Fig. 2: The terminal user interface that represents the current network state (original network) as it describes nodal information for navigational reference to create a user-friendly environment during observations.

Once the user has finished their observations of the initial network, the program will simulate a power outage, choosing a random selection of nodes and links to shutdown due to network failure. The program will then update the GUI accordingly to represent the current state of the network, while illustrating the optimal path and distance of each connected node in the terminal window. Faulted nodes will be removed from the network, deleting all of the target's connections

and deallocating the node's attributes effectively. Similarly, faulted links will be removed from the network, leaving the connected nodes untouched. If the removal of a faulty attribute creates an accessibility issue in which a node can no longer be accessed by the source node, then the graph will highlight the inaccessible node in red, illustrating an inaccessibility message in the terminal window. Figure 3 illustrates the updated network GUI after the first failure interval due to a power outage.

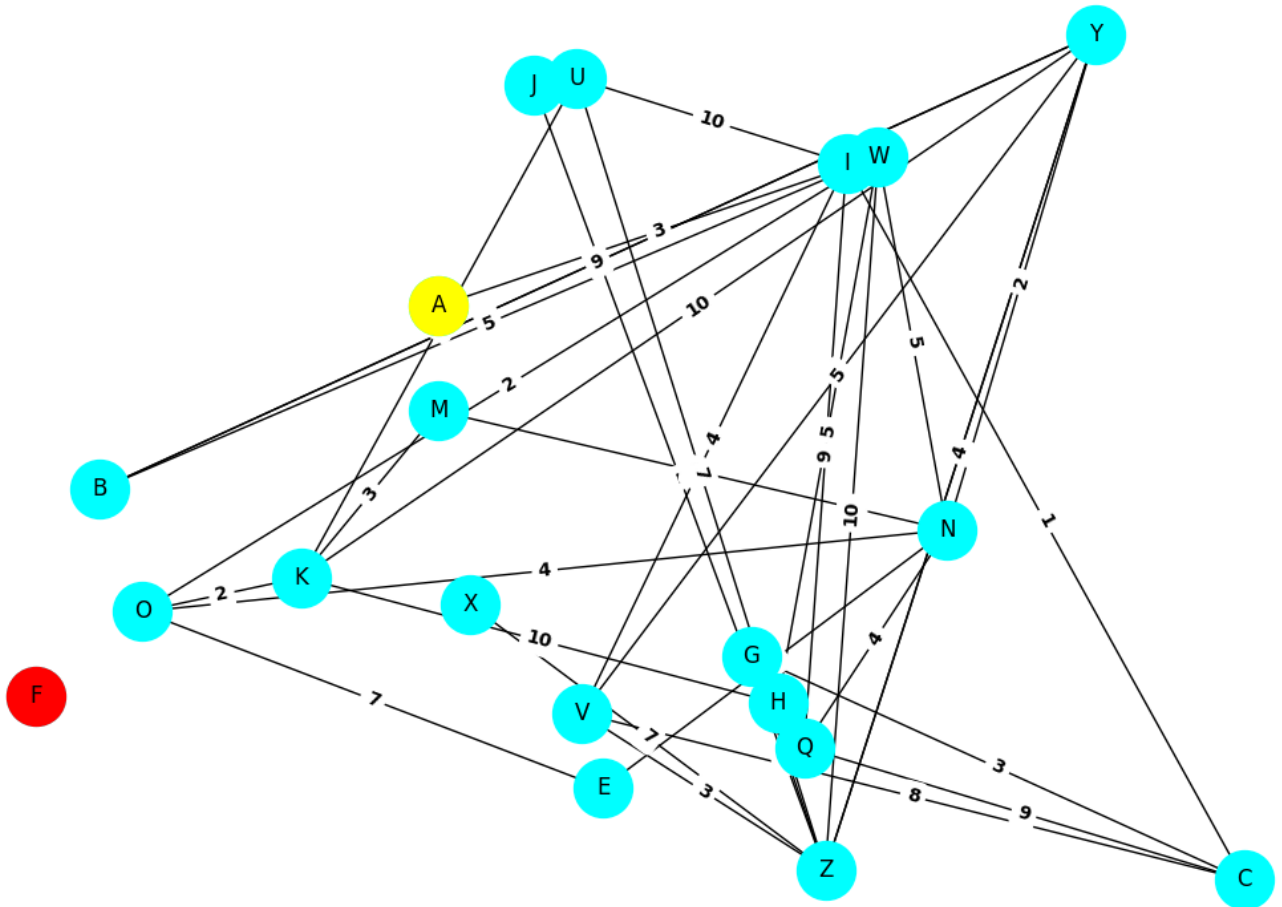


Fig. 3: An updated state of the bidirectional weighted network after the first faulty instance. The simulation removed nodes ['L', 'T', 'S', 'D', 'P', 'R'] and links [('R', 'X'), ('B', 'X'), ('F', 'I'), ('P', 'R'), ('O', 'Y'), ('N', 'W')] from the original network due to router/connection failure. Unreachable nodes highlighted in red signify isolation from the source node as they no longer remain in the network.

Similar to the initial graphical representation, the user will utilize the terminal window for additional information. For each faulty instance, the terminal will provide a graphical analysis for each new network state (i.e., network statistics, error messages, and Dijkstra's Algorithm).

Figure 4 showcases the terminal's graphical analysis of the updated GUI environment after the first round of network failures.

```

////////////////////
//// Faulty Network ////
////////////////////

ATTENTION: POWER OUTAGE INCOMING. SOME NETWORKS HAVE GONE OFFLINE. SOME NETWORKS MAY FAIL INTERMITTENTLY!

There has been a fault in node(s): ['L', 'T', 'S', 'D', 'P', 'R']! Removing faulty networks immediately!

There has been a fault in edge(s): [('R', 'X'), ('B', 'X'), ('F', 'I'), ('P', 'R'), ('O', 'Y'), ('N', 'W')]! Removing faulty edges immediately!

////////////////////
//// Network Information ////
////////////////////

Number of nodes: 20

Online Network Nodes: ['A', 'B', 'C', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'M', 'N', 'O', 'Q', 'U', 'V', 'W', 'X', 'Y', 'Z']

Number of edges: 36

Current Edge Pairings: [('A', 'W', {'weight': 3}), ('B', 'Y', {'weight': 8}), ('B', 'Y', {'weight': 9}), ('B', 'W', {'weight': 5}), ('C', 'Q', {'weight': 9}), ('C', 'G', {'weight': 3}), ('K', {'weight': 10}), ('H', 'W', {'weight': 5}), ('I', 'W', {'weight': 4}), ('I', 'Q', {'weight': 9}), ('I', 'U', {'weight': 10}), ('I', 'V', {'weight': 4}), ('J', 'U', {'weight': 3}), ('J', 'Z', {'weight': 4}), ('N', 'W', {'weight': 5}), ('N', 'O', {'weight': 4}), ('N', 'Q', {'weight': 4}), ('N', 'Y', {'weight': 2}), ('O', 'W', {'weight': 2}), ('U', 'Z', {'weight': 7}), ('V', 'Z', {'weight': 4})]

////////////////////
//// Dijkstra's Algorithm w/ Online Networks ////
////////////////////

A to A (path cost = 0): ['A']
A to B (path cost = 8): ['A', 'W', 'B']
A to C (path cost = 8): ['A', 'W', 'I', 'C']
A to E (path cost = 12): ['A', 'W', 'O', 'E']
A to F (path cost = unavailable): Node F is unreachable.
A to G (path cost = 11): ['A', 'W', 'I', 'C', 'G']
A to H (path cost = 8): ['A', 'W', 'H']
A to I (path cost = 7): ['A', 'W', 'I']
A to J (path cost = 15): ['A', 'W', 'Z', 'J']
A to K (path cost = 7): ['A', 'W', 'O', 'K']
A to M (path cost = 10): ['A', 'W', 'O', 'K', 'M']
A to N (path cost = 8): ['A', 'W', 'N']
A to O (path cost = 5): ['A', 'W', 'O']
A to Q (path cost = 12): ['A', 'W', 'N', 'Q']
A to U (path cost = 14): ['A', 'W', 'O', 'K', 'U']
A to V (path cost = 11): ['A', 'W', 'I', 'V']
A to W (path cost = 3): ['A', 'W']
A to X (path cost = 20): ['A', 'W', 'Z', 'X']
A to Y (path cost = 10): ['A', 'W', 'N', 'Y']
A to Z (path cost = 13): ['A', 'W', 'Z']

* Creating network interface *

* Network interface has been downloaded for your convenience *

```

Fig. 4: The terminal user interface that represents the current network state (network after first faulty instance) as it describes nodal information for navigational reference to create a user-friendly environment during observations.

As indicated by the global variable representing the total number of faulty instances, the cycle repeats “x” number times. For each instance, the GUI and terminal prompts are updated appropriately until the end of the simulation. Figure 5 illustrates the current state of the the network GUI after three intermittent faulty instances.

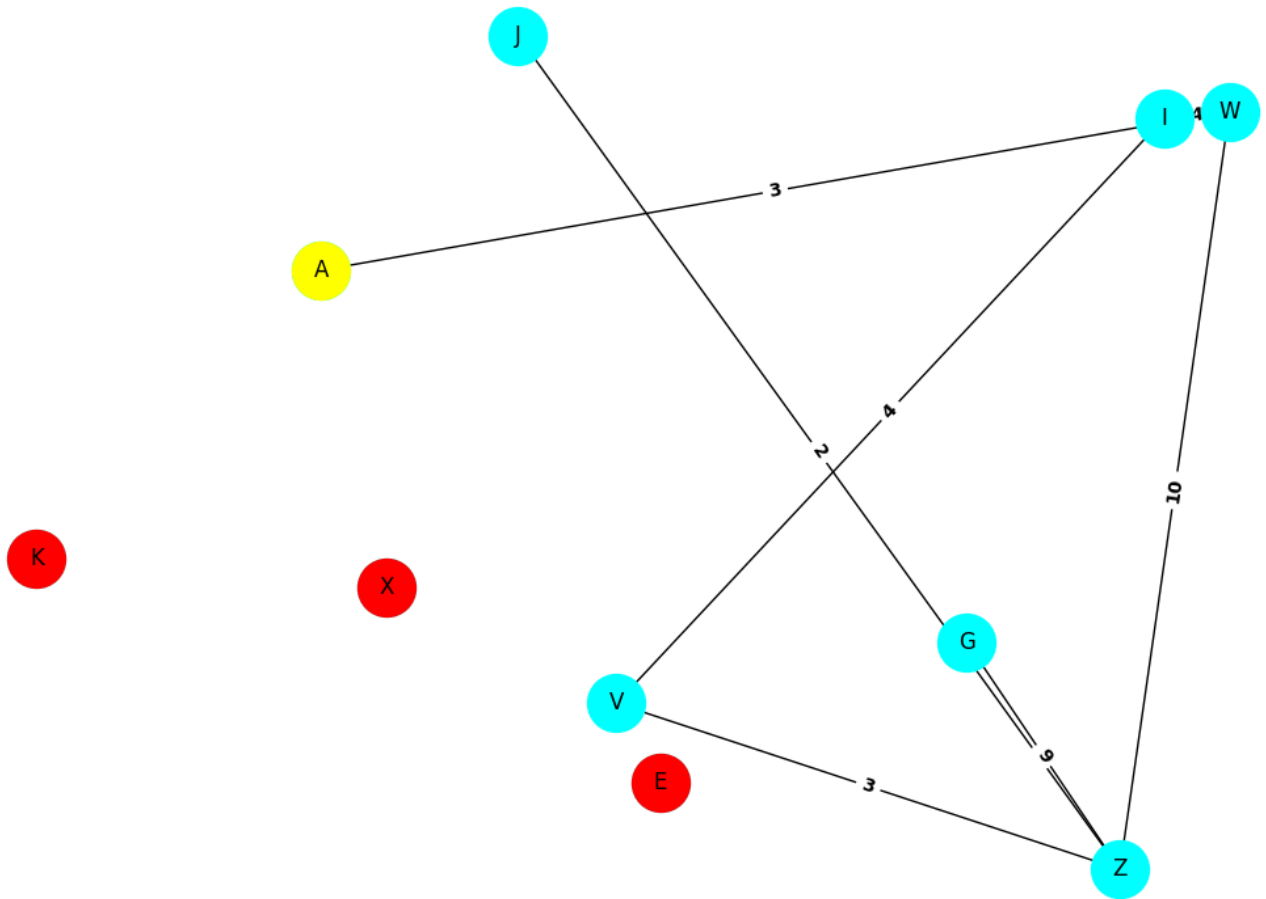


Fig. 5: An updated network state of the bidirectional weighted graph after three faulty instances.

Similar to the previous graphical simulations, Fig. 6 illustrates the terminal’s graphical analysis of the updated GUI environment after three intermittent faulty instances.

```
-----
ATTENTION: MORE NETWORKS FAILING. NETWORK INTERFACE WILL BE UPDATED IN 3 SECONDS.
```

```
There has been a fault in node(s): ['F', 'N', 'H', 'M', 'O', 'B']! Removing faulty networks immediately!
```

```
There has been a fault in edge(s): [('E', 'O')]! Removing faulty edges immediately!
```

```
////////////////////
//// Network Information ////
////////////////////
```

```
Number of nodes: 10
```

```
Online Network Nodes: ['A', 'E', 'G', 'I', 'J', 'K', 'V', 'W', 'X', 'Z']
```

```
Number of edges: 7
```

```
Current Edge Pairings: [('A', 'W', {'weight': 3}), ('G', 'Z', {'weight': 9}), ('I', 'W', {'weight': 4}), ('I', 'V', {'weight': 4}), ('J', 'Z', {'weight': 2}), ('V', 'Z', {'weight': 2})]
```

```
////////////////////
//// Dijkstra's Algorithm w/ Online Networks ////
////////////////////
```

```
A to A (path cost = 0): ['A']
A to E (path cost = unavailable): Node E is unreachable.
A to G (path cost = 22): ['A', 'W', 'Z', 'G']
A to I (path cost = 7): ['A', 'W', 'I']
A to J (path cost = 15): ['A', 'W', 'Z', 'J']
A to K (path cost = unavailable): Node K is unreachable.
A to V (path cost = 11): ['A', 'W', 'I', 'V']
A to W (path cost = 3): ['A', 'W']
A to X (path cost = unavailable): Node X is unreachable.
A to Z (path cost = 13): ['A', 'W', 'Z']
```

```
* Creating network interface *
```

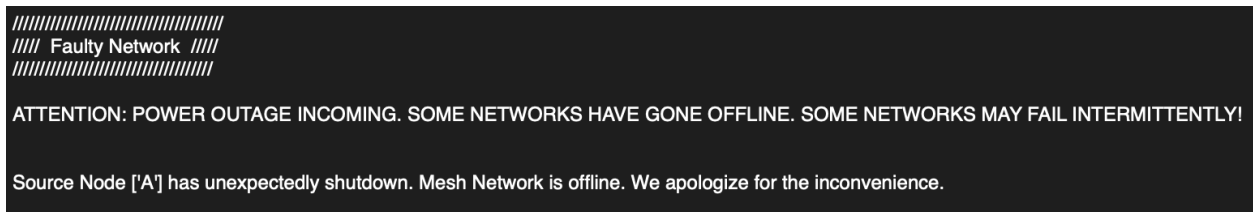
```
* Network interface has been downloaded for your convenience *
```

Fig. 6: The terminal user interface that represents the current network state (network after three faulty instances) as it describes nodal information for navigational reference to create a user-friendly environment during observations.

After executing a predetermined number of faulty instances, the program will simulate a nodal recovery instance. The terminal will indicate that the power is being restored, allowing the first set of faulty nodes to reactivate. Using queues to store faulty node attributes, the recovered nodes will be added back into the network with their corresponding links — updating the GUI accordingly to present the recovered network state, where recovered nodes are highlighted in lime green. It is important to note that if a recovered node is unreachable via source node pathing, then the program will highlight the node in red to signify isolation. Figure 7 illustrates the updated network GUI after nodal recovery due to power restoration.

After conducting a nodal recovery scenario, the program will prompt the user that the simulation is now complete, saving all network diagrams into the executable's directory for future reference.

Given the program's high degree of detail and complexity, the group has integrated multiple error handling mechanisms to ensure a stable and universal simulation. Through the use of global variables, users are able to modify the program values (e.g., number of nodes, source node, number of faults, and probability) to simulate different scenarios - promoting the simulation's scalability and accessibility. Using graphical representations to aid users in visualizing the network map, the terminal acts as a guide as it highlights network behaviors. Along with displaying relevant information to the user interface, the system saves all network diagrams to the user's machine for further evaluation. During all cases of nodal failure, the system will remove the node and its corresponding links in order to rescale the network to prevent resource depletion. If the removal of a node/link creates an inaccessible node, the program will register its isolation, recolor the node in the graph, and inform the user that the router is no longer connected to the source/domain. As seen in Fig. 9, if the source node is subjected to network failure due to energy and/or traffic complications, the simulation will immediately end, informing the user that the network is no longer active due to a force shutdown. By ensuring that all nodes are susceptible to failure, the group promoted the integrity of the simulation by mirroring potential issues and addressing their potential responses.

A terminal window with a black background and white text. The text is as follows:

```
////////////////////  
//// Faulty Network ////  
////////////////////  
  
ATTENTION: POWER OUTAGE INCOMING. SOME NETWORKS HAVE GONE OFFLINE. SOME NETWORKS MAY FAIL INTERMITTENTLY!  
  
Source Node ['A'] has unexpectedly shutdown. Mesh Network is offline. We apologize for the inconvenience.
```

Fig. 9: The terminal demonstrates an example of error handling by illustrating a scenario in which the source node ['A'] is subjected to nodal failure, shutting down the entire mesh network as there no longer exists a starting transmission. Ensuring all possibilities of nodal failure promotes real-life behavior and simulation integrity.

After creating error handlings that protect the network's core functionality, the group added some baseline conditions to ensure proper syntax handling. In particular, the group added an if statement to ensure that the user does not assign the total number of network nodes to any value below 1 as it would result in a simulation error as mesh networks cannot be produced without an existing router. Given that source nodes are a mesh network requirement, the group implemented an if statement checking for a source node input. If the user did not assign an initial character to the source node, then the system would automatically generate one from the ASCII Uppercase

library to conduct the simulation. Furthermore, the group implemented a conditional if statement during the network initialization process to check that the indicated source node exists in the original node list. If the source node is unavailable during initialization, then the network cannot be generated as there does not exist a starting transmission. In addition, the group also implemented an if statement to differentiate the original network and faulty network by referencing the total number of faults. If the user assigns a value greater than 0, then the program will simulate a faulty network scenario as there exists a number of faulty instances. However, if the value is less than or equal to 0, then the user is requesting for a regular network generator with no faulty nodes/links.

Out of the Box Thinking: A Novel Contribution

In relation to novel contributions, the goal of the group is to make things easier for the user in order to mimic real life scenarios as much as possible. For example, a graphical interface is provided that shows the physical network with color coded nodes. This promotes a user friendly environment by creating a visual map to navigate through the simulated network — promoting program accessibility. In each network figure, yellow represents the source node, red represents unreachable nodes, and lime represents recovered nodes. A recovery scenario is implemented to add additional project functionality by having nodes reactivate after faulting — creating a more realistic scenario by accounting for power restoration. In addition, each network instance is saved to the user's machine to use for future reference. In every figure, the terminal will print out all statistical information for the network. This includes information such as the number of nodes, the number of edges, as well as the number of edge pairings.

Another novel feature implemented is the randomness of the code. Every time the code runs, the result will be something different. This allows for the user to use the code multiple times and analyze different results. In addition, the code has global variables that can be adjusted to a user's preference. Examples of this could be the number of total simulated faults or total number of nodes in the network. These are the novel contributions that the group implemented.

Results and Analysis

The goal of the simulation is to have nodes and links fail intermittently with each node having a probability to fail and adapt to avoid the faulty node. After many run throughs of the program, the team found that the simulation accomplished all of the functionalities required by the project. These are having nodes and links failing intermittently, and adapting and re-routing to avoid the

faulty link and node as seen in Fig. 3, Fig. 5, and Fig. 7. The results that we got were as expected for how the simulation should behave. In the case of the source node failing, the network would go online as seen previously in Fig. 9. In all other cases, nodes would have a probability of failing. Therefore, accomplishing the failing intermittently goal. Additionally, the team added node recovery functionality and the results were as expected. After the node and link failures, some nodes that previously failed would be added to the network and re-routed as seen in Fig. 7.

Conclusion

The project simulated a mesh network where nodes and links may fail. The nodes and links fail intermittently with each node and link having a certain probability to fail. When the failure occurs, the network adapts and re-routes to avoid the faulty node or link. The team additionally added node recovery functionality to the simulation by allowing some nodes to be added back to the network after the failures occur and re-routing the nodes so that the newly recovered working nodes are fully connected to the network.

References

Mesh Networks (website). Retrieved from

<https://cis-india.org/telecom/resources/mesh-networks#:~:text=By%20Definition%2C%20Mesh%20networks%20are,to%20be%20live%20and%20healthy>.

Open Shortest Path First (OSPF) (website). Retrieved from

<https://www.metaswitch.com/knowledge-center/reference/what-is-open-shortest-path-first-ospf>