Names: Blong Vang, Clint Vega
For: CPE400 Final Report Spring 2022
Project #6: Dynamic routing mechanism design in faulty network
Github: https://github.com/Vang-Blong/CPE-400-Final-Project

Functionality

The functionality of our program was to simulate a network in which nodes or edges may fail. Given these constraints, the network should automatically generate a new set of routing tables for shortest paths each time a node or edge fails. Additionally, we added attributes that would allow us to reverse calculate the fail rate of an edge over time. This feature could be used to further contribute against the weight of an edge or the trust score of a node on a network.

Our team was able to successfully implement and deliver the following with our program:

- Generate a random network for simulating on to prove program flexibility
- Attribute fail rates to nodes and edges
- Properly reroute a network when an edge fails, using Dijkstra's Algorithm
- Reverse calculate fail rate over time to contribute toward an edge's total weight, or 'score' attribute
- Display reverse-calculated fail rate for each edge using Matplotlib
- Provide runtime metrics usable in further data analysis

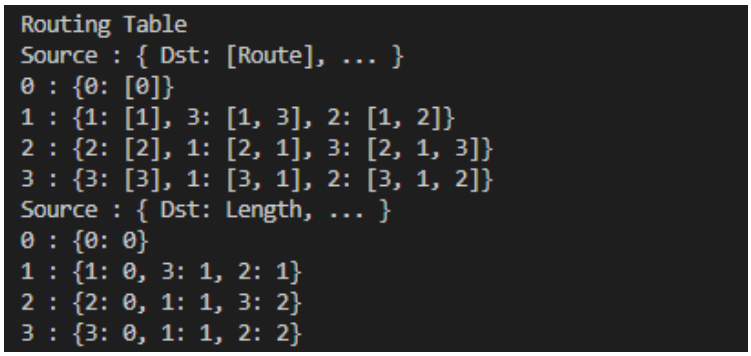

Figure 1. In the event of node failure, Dijkstra's Algorithm is recalculated and a new routing table is generated.

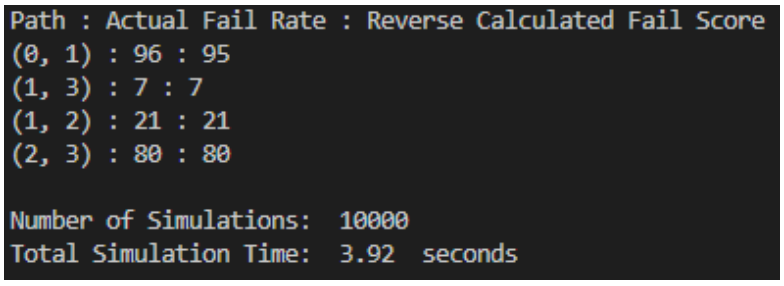

Figure 2. After all simulations are complete, the actual fail rate of a path is compared to the reverse calculated fail rate score. Runtime and the number of simulations are also presented.

Novel Contribution

For our novel contribution, our team focused on network connectivity, inspired by the definition of a "trust score" in social and business networks in a Google Patent on network trust scores [Chan]. Whereas computer networking trust scores are mostly security-focused, fault tolerance is not considered a major metric [Buchanan]. For measuring network connectivity, we used a "score", or "fail rate score" calculated following a formula mentioned by and implemented in python [Agarap].

The actual probability of failure rate of a computer network's components would not be known before being brought online. As such, our novel contribution is reverse engineering the failure rate of a computer network over time. This metric could be used in trust scores, fault tolerance calculations, or other statistic-driven applications. Most importantly, this metric could be used for determining the routing of network traffic before failures occur.

This formula we attempted to emulate mathematically is as pictured below [Jiang]:

Algorithm 1 Estimating α-high-density-set

Parameters: α (density threshold), k.
Inputs: Sample points $X := \{x_1, .., x_n\}$ drawn from f.
Define k-NN radius $r_k(x) := \inf\{r > 0 : |B(x, r) \cap X| \geq k\}$ and let $\varepsilon := \inf\{r > 0 : |\{x \in X : r_k(x) > r\}| \leq \alpha \cdot n\}$.
return $\widehat{H}_\alpha(f) := \{x \in X : r_k(x) \leq \varepsilon\}$.

Algorithm 2 Trust Score

Parameters: α (density threshold), k.
Input: Classifier $h : \mathcal{X} \to \mathcal{Y}$. Training data $(x_1, y_1), ..., (x_n, y_n)$. Test example x.
For each $\ell \in \mathcal{Y}$, let $\widehat{H}_\alpha(f_\ell)$ be the output of Algorithm 1 with parameters α, k and sample points $\{x_j : 1 \leq j \leq n, y_j = \ell\}$. Then, return the trust score, defined as:

$$\xi(h, x) := d\left(x, \widehat{H}_\alpha(f_{\widetilde{h}(x)})\right) / d\left(x, \widehat{H}_\alpha(f_{h(x)})\right),$$

where $\widetilde{h}(x) = \text{argmin}_{l \in \mathcal{Y}, l \neq h(x)} d\left(x, \widehat{H}_\alpha(f_l)\right)$.

Figure 3. Given a high density set of information, we can determine a trust score through simple division of respective sample points against the original data set [Jiang].

Our own practical implementation using Python was as follows:

```
#get score by dividing number of successes by instances ran
current_up = nx.get_edge_attributes(G,'up_count')
current_score = 100 - int(current_up[(edge1,edge2)]) / counter * 100
```

Fig 4. Maintain the total number of times an edge is in the "UP" or "online" state as a unique attribute to that edge, and then divide it out by the number of simulations run by the user. We then invert this score on a scale of 0-100 to align with the actual failure rate of an edge or node.

We can then visualize this attribute in a graph:

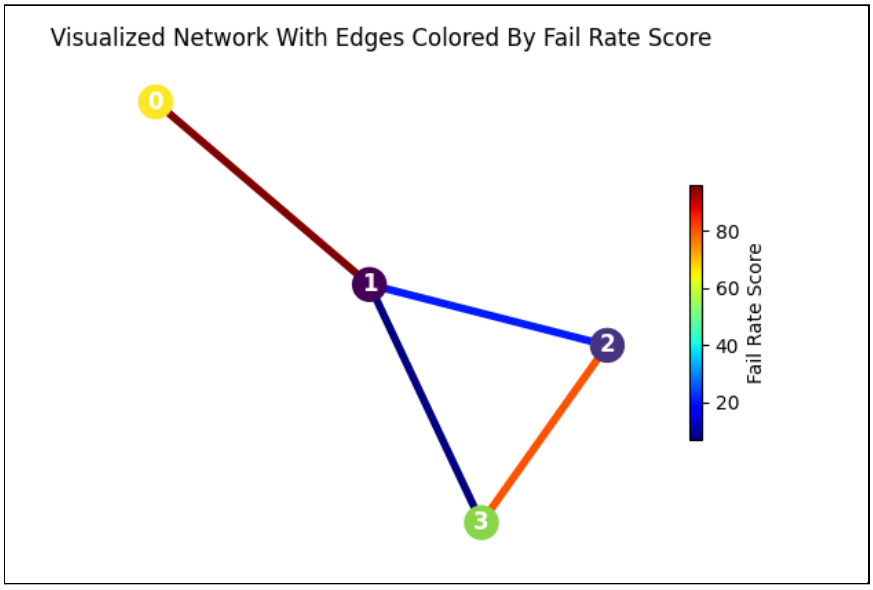

Fig 5. The higher the "Fail Rate Score" of a node is, the closer to the color red it becomes. This particular graph was generated after running the simulation for 1000 instances. A high Fail Rate Score can contribute negatively to the weight or trustworthiness of a network, whereas a low Fail Rate Score has minimal effect.

When only running a single instance of a simulation, we instead get results as pictured below:

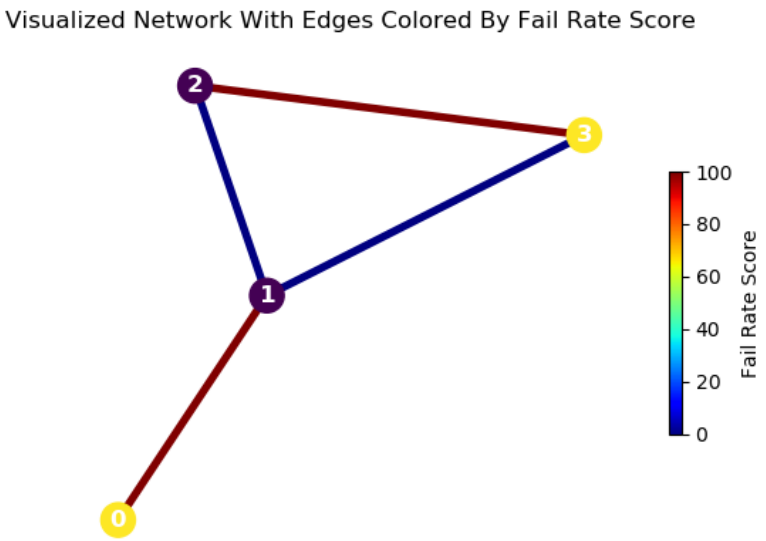

Fig 6. Note how the "Fail Rate Score", while indicative of which edges are active after a single simulation, is not actually useful in determining the long-term viability or performance of the network.

Although we did not go further with the applications of the metric of our “Fail Rate Score”, we believe it can be applied toward ensuring the integrity of the network and potentially many other applications. Instead of simply rerouting a network every time a node fails, we can simply use the reverse-engineered fail rate to determine whether or not to send packets along that edge or node. If high fault tolerance exists, a high fail rate edge or node can be used, whereas a low Fail Rate node or edge should be utilized if a network packet cannot be dropped.

Results & Analysis

We were able to successfully reroute the paths of a network using Dijkstra’s Algorithm in the event of node failure. When nodes were no longer considered to be in a failed state, they were calculated in subsequent simulations, to accurately route the network as well as reverse calculate the failure fail score of a given edge. We were successful in accurately reverse calculating the fail score within a single point margin of error. We were able to maintain the accuracy of our calculations as well as successful rerouting for up to one million simulations.

Our calculations benefit from long-term data as they rely on gathering information about a path’s failure rate. This can be noted in the following figures:

```
Path : Actual Fail Rate : Reverse Calculated Fail Score
(0, 1) : 96 : 80
(1, 3) : 7 : 10
(1, 2) : 21 : 30
(2, 3) : 80 : 70

Number of Simulations:  10
Total Simulation Time:  0.01  seconds
```

```
Path : Actual Fail Rate : Reverse Calculated Fail Score
(0, 1) : 96 : 95
(1, 3) : 7 : 7
(1, 2) : 21 : 21
(2, 3) : 80 : 80

Number of Simulations:  10000
Total Simulation Time:  3.92  seconds
```

Fig 7. Note how the run of ten-thousand simulations was able to reverse calculate the fail score within one point, while when only given ten simulations, there were differences of up to ten points between the reverse calculated fail score and the actual fail score.

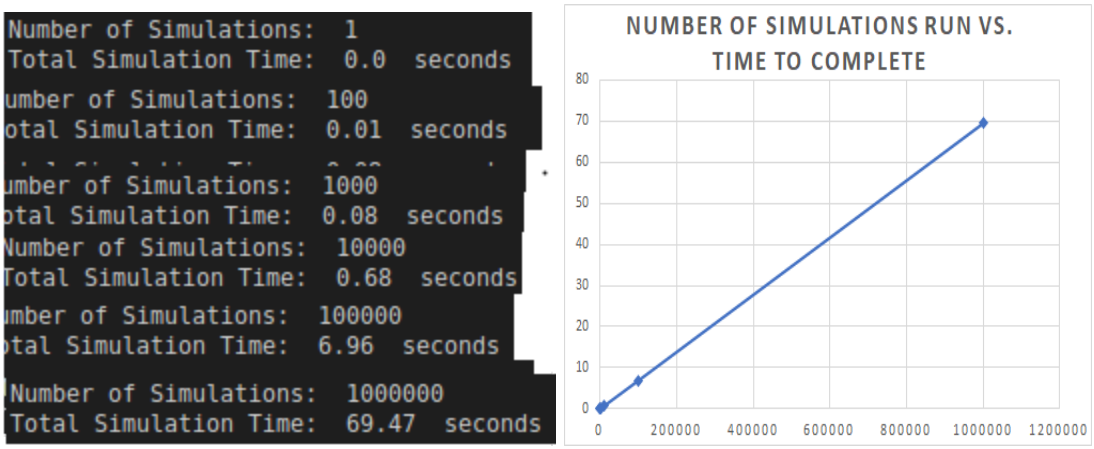

Fig 8. Note how runtime linearly scales with the number of simulations run, despite having a variable number of failures in the network.

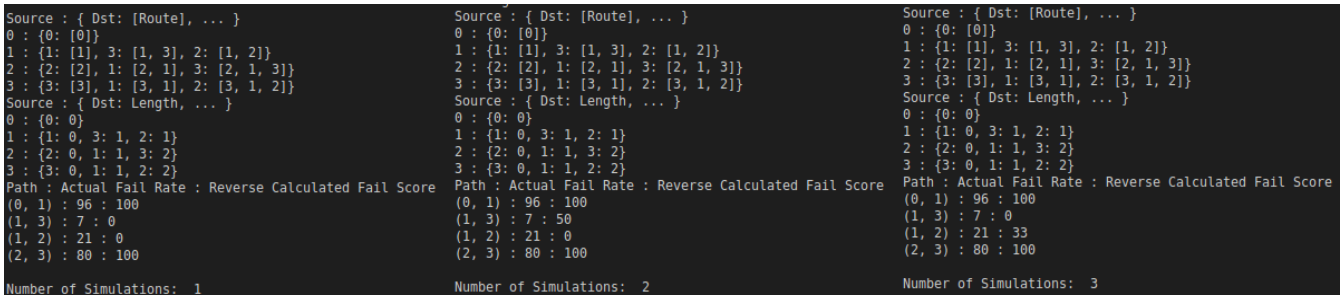

Fig 9. Note how even with gradual incrementation of simulations, the failures may stay consistently the same throughout, but ultimately the reverse calculated “Fail Rate Score” metric will still account for this and change over time.

In conclusion, we were able to implement a network that could immediately calculate a new shortest-path routing table in the instance of a network failure using Dijkstra’s Algorithm, and also calculate the failure rate over time. This metric enables us to account for failure when routing network traffic over an extended period of time.

References

Agarap, Abien Fred. “How Can I Trust You?” *Medium*, Medium, 3 Apr. 2020, https://medium.com/@afagarap/how-can-i-trust-you-fb433a06256c.

Buchanan, Justin. “Trust Analytics and Anti-Spoofing Protection: It's Already in Your Network.” *Cisco Blogs*, 20 July 2021, https://blogs.cisco.com/networking/trust-analytics-and-anti-spoofing-protection-its-already-in-your-network.

Chan, Leo; Chrapko, Evan; Mawji, Ashif; Chrapko, Shane, and Marsh, Stephen.“US9578043B2 - Calculating a Trust Score.” *Google Patents*, Google, https://patents.google.com/patent/US9578043B2/en.

Jiang, et al. *To Trust or Not to Trust a Classifier - Arxiv*. https://arxiv.org/pdf/1805.11783.pdf.