Simulating Smart Grid Cascade Failure Mitigation

Claudia Whitten
Department of Computer Science
California Institute of Technology
Pasadena, CA 91126
whitten@caltech.edu

ABSTRACT

Smart grid power systems have the potential to reduce energy usage by modernizing current electrical grids. These systems work by allowing real-time monitoring and adjustment of power flow using communication between net metering systems located at the sites of both the suppliers and consumers. Smart grids must mitigate the effects of disruptions within the network by appropriately adjusting power flow routes and preventing cascading failure propagation. The need for strict network security is one of the reasons that widespread implementation of smart grids has yet to occur. In order to better understand the manner in which such issues can be resolved, I am working to design an application that allows users to implement and test cascade failure mitigation algorithms.

1. INTRODUCTION

Modern power plants rely on a variety of resources to generate electricity. These include non-renewable resources like fossil fuels which contribute to environmental issues such as global warming when they release carbon dioxide as a result of being burned. The recent initiative to become more environmentally aware and energy independent has led to the development of smart grid technology [1]. These smart grid systems are a modernization of current electrical grids, utilizing a similar physical infrastructure and monitoring system as their predecessors, but responding more intelligently to changes within the network [2]. An ideal smart grid system would increase the security, reliability, and efficiency of electricity networks and be able to perform tasks such as scheduling different household appliances to run at off-peak demand hours and track the energy usage of each appliance in order to lower total energy consumption.

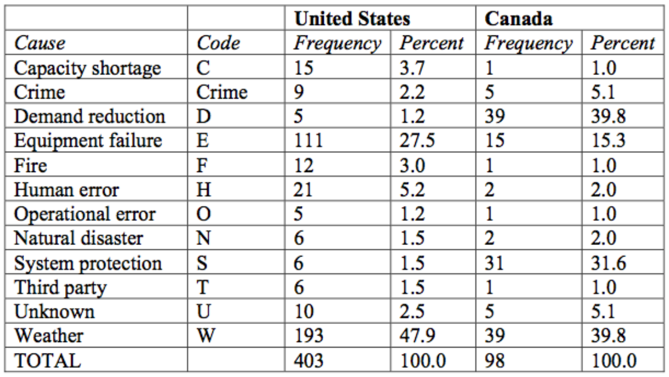
However, the design and implementation of smart grid systems has proven to be difficult. There are many unresolved issues that have limited the implementation of these improved networks, not the least of which is the need for better cyber security. In order to safely transition from the current power grid to a smart grid system, current cyber security protocols need to be updated and improved [3]. One such security vulnerability is the effect of disruptions within the network that cause cascading failures, which have the potential to bring down entire electrical grids if they are not effectively mitigated [4, 5]. That is, a blackout that occurs at one power station has the potential to shift additional load to other nearby power plants, causing further blackouts due to the increased load exceeding the permitted capacity. These incidents can be caused by a variety of factors (see figure 1), but each outage has the potential to propagate regardless of the cause (though this can be used to predict the characteristics and magnitude of the outage). By resolving such security issues, smart grid systems will be able to be safely implemented, resulting in a more reliable, cost effective, and environmentally friendly electricity network.

2. SMART GRID GUI

In order to help the user design and simulate different smart grid network configurations, I have designed a graphical user interface (henceforth known as the Smart Grid GUI). If properly implemented, the Smart Grid GUI would allow users to perform the following tasks:

- Specify the network topology along with points of failure in one of two ways.
    1. Describing the network topology and disruptions in a file formatted for interpretation by the Smart Grid GUI.
    2. Selecting options from the Smart Grid GUI menus and toolbars.
- See the network topology.
- Select a cascading failure mitigation algorithm.
- Run simulations to determine the effectiveness of different algorithms.
- Aggregate and compare statistics from simulations.

I have begun the task of implementing the Smart Grid GUI and have included several figures (described later in the paper) to demonstrate how it could possibly look when nearer to completion. After initially attempting to design the GUI using Java Swing without the help of an integrated development environment (IDE), I chose to use the NetBeans IDE 7.0 Swing GUI Builder to aid in the design of the user interface.



		United States		Canada	
Cause	*Code*	*Frequency*	*Percent*	*Frequency*	*Percent*
Capacity shortage	C	15	3.7	1	1.0
Crime	Crime	9	2.2	5	5.1
Demand reduction	D	5	1.2	39	39.8
Equipment failure	E	111	27.5	15	15.3
Fire	F	12	3.0	1	1.0
Human error	H	21	5.2	2	2.0
Operational error	O	5	1.2	1	1.0
Natural disaster	N	6	1.5	2	2.0
System protection	S	6	1.5	31	31.6
Third party	T	6	1.5	1	1.0
Unknown	U	10	2.5	5	5.1
Weather	W	193	47.9	39	39.8
TOTAL		403	100.0	98	100.0

Figure 1: Distribution of primary causes of outages in the U.S. and Canada between 1990 and 2004. [5]

2.1 Design

The Smart Grid GUI is designed to be easily extensible with respect to future implementation of other algorithms that mitigate cascade failure or perform other operations on given network topologies. This application includes classes that represent power stations (nodes) and voltage lines (links). These classes have the potential to be extended and subclassed in the event that a user has more specific needs, such as representing a particular type of power station. Any cascading failure mitigation algorithm implemented either by myself or another user can be contained within another class or set of classes. These classes can take as arguments the network topology and points of failure within the network, and can return the means by which the cascading failure is addressed. For example, the class which implements the selective node removal algorithm can return the set of nodes that should be removed from the network in order to limit the propagation of the failure, along with the timing of the removals.

2.2 Implementation

This application is written in Java. The object-oriented nature of the language lends itself to this project, and I have taken care to ensure that separate components are appropriately encapsulated. I have made extensive use of Swing, which is Java's primary widget toolkit. Initially, I implemented separate classes to represent power stations, voltage lines, and potential mitigation algorithms; however, upon my discovery of JGraphT, a free Java graph-theory library created by Barak Naveh and other contributors, I attempted to redesign the application around this library. Ease of use is a particularly important component of this project, since I feel that it is important to be able to customize and visualize the simulated power grid network. For that reason, I would continue to use the NetBeans GUI Builder for further work on this project, and would suggest that anyone else working on the project to do the same.

2.3 Current Progress

Currently, the project is incomplete with respect to what I had hoped to accomplish over the course of the term. I have included several figures later in this report that show what I have already designed and implemented, along with describing what future work on the project would likely yield in terms of application design and structure. In the next section, I will go into extensive detail about a selective node removal algorithm that I have not yet completely implemented; however, this detail with respect to potential class structures should allow future completion of this work to progress rapidly.

3. DISRUPTION MITIGATION ALGORITHMS

There are three disruption mitigation algorithms that I initially intended to implement as part of the Smart Grid GUI. These include a weighted least squares algorithm, a mixed-integer optimization algorithm [7], and a selective node removal algorithm [6]. I have made substantial progress toward completing the implementation of the selective node removal algorithm. However, time did not permit me to complete the implementation of the weighted least squares algorithm and mixed-integer optimization algorithm. For the sake of completeness, I will briefly describe them here before going into detail concerning the selective node removal algorithm.

3.1 Weighted Least Squares Algorithm

My first attempt at implementing a cascading failure mitigation algorithm was designing of a weighted least squares algorithm; however, I remain unconvinced that my attempts were correct. Thus, I will give some background related to the theory behind this algorithm, then will later discuss the selective node removal algorithm with which I have had more success. The least squares method attempts to find an approximate solution to a set of equations where there are more equations than unknowns. This solution minimizes the sum of squared residuals. The weighted least squares method assigns a weight to each observation that indicates the uncertainty of the measurement, and is generally a function of the variance of the data. These systems of equations are often solved using iterative methods, linearly approximating then subsequently refining the approximation.

Attackers who feed bad information into a power grid system can overload a particular power plant and cause a blackout that could propagate throughout the system. Thus a weighted least squares approach that quickly determines the actual power flow through the network and shifts load accordingly could mitigate this situation. Once an application has determined, based on measurements taken at different points in the system, what the current load is on different nodes, it could recalculate the best way to redistribute the load so as to not exceed the nodes' capacity.

3.2 Mixed-Integer Optimization Algorithm

I have not yet implemented this algorithm in the Smart Grid GUI; however, for the sake of completeness, I will give a description here. This algorithm is motivated by the need to take into account the many steps involved in a cascading failure when determining an optimal algorithm to control the damage done to the network. It is formulated as a multi-stage mixed-integer programming problem that is designed to terminate the propagation of the cascade after a certain number of cycles while satisfying as much demand as possible. The result is an optimal schedule for demand shedding when given a deterministic set of power line outages.

This algorithm was the most difficult to understand of the three I examined, due in part to the length of the paper which described it. It is defined as an affine, adaptive, distributive control algorithm that is computed when the initial failure occurs, and then is subsequently deployed. One potential problem with such an approach is that this requires a large amount of time to compute immediately after the failure, which might not work well in practice depending on how quickly the blackout propagates. However, once this calculation at time zero is complete, no further computation is needed; the algorithm is simply applied to the network. The author describes a general cascade control template given that there is a time R after which no power lines can have load exceeding their capacity, described below:

Input: a power grid with graph G. Set $G^1 = G$.

Step 0. Compute control algorithm.

For $r = 1, 2, ..., R-1$, **do**

1. Set f^r = vector of power flows in G^r.
2. **Observe** state of grid (from state estimation).
3. **Apply** control.
4. Set g^r = vector of resulting power flows in G^r.
5. Set O^r = set of lines of G^r that become outaged in round r.
6. Set $G^{r+1} = G^r - O^r$. Adjust loads and generation in G^r.

Termination (round R). If any island of G^R has line overloads, proportionally shed demand in that island until all line overloads are eliminated.

The author of the paper also describes an affine control policy where round R is again the time at which the cascade must terminate. I include this policy, below, noting that the term *bus* is simply another word for node:

Input: a power grid with graph G (post-initiating event). Set $G^1 = G$.

0. Compute triples (c_v^r, b_v^r, s_v^r) for each $r < R$ and v.

For $r = 1, 2, ..., R-1$, **do**
(comment: controlled round r of the cascade)

1. Set f^r = vector of power flows in G^r, and d_v^r = the demand of any bus v.
2. For any demand bus v, let κ_v^r be its data observation. **Apply control:** if $\kappa_v^r > c_v^r$, reset the demand of v to
$$min\{1, [b_v^r + s_v^r(c^r - \kappa_v^r)]^+\} d_v^r. \quad (1)$$
3. Adjust generator outputs in each component of G^r so as to match demand.
4. Set O^r = set of lines of G^r that become outaged as a result of the flows instantiated in Step 4.
5. Set $G^{r+1} = G^r - O^r$. Adjust demands and supplies in G^r.

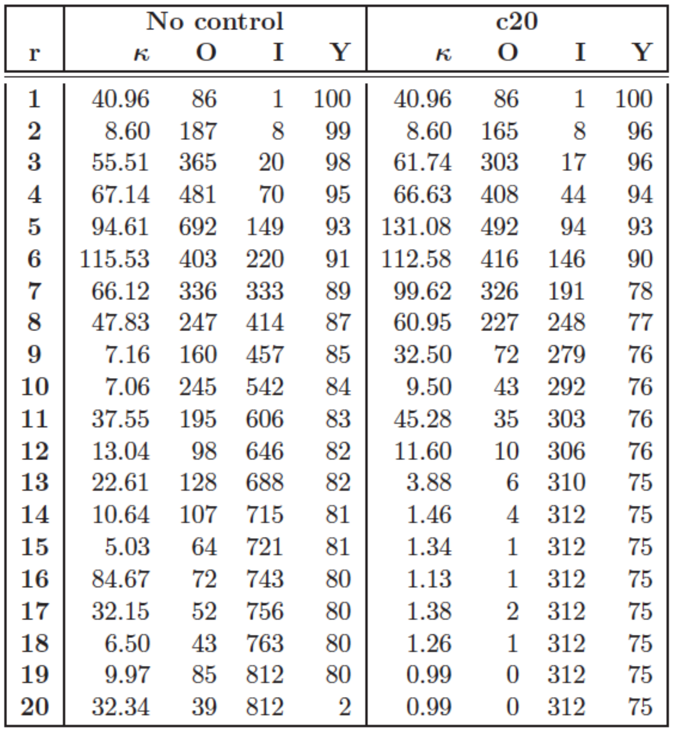
Round R. For any component K of G^R, set
$$\Psi_K^R \doteq min\{1, max_{j \in K}\{|f_j^R|/u_j\}\}. \quad (2)$$

If $\Psi_K^R > 1$, then any bus v of K resets its demand to d_v^R / Ψ_K^R.

The author then presents experimental results that used data from the U.S. Eastern Interconnect system, which contains approximately 15,000 buses, 23,000 lines, 2,000 generators, and 6,000 load buses. The optimal control, henceforth referred to as **c20** was computed where

1. $c_v^r = b_v^r = 1$ for all v and r.
2. $s_v^r = 0$ for all v and $10 < r$.
3. For each $1 \leq r \leq 10$, either $s_v^r = 0.005$ for all v or $S_v^r = 0$ for all v.

The case was considered where two lines were removed ($K = 2$) and $R = 20$ rounds. Figure 2 describes the differences in results between **c20** and no control for this case for each round $r < 20$. κ is the maximum line overload at the beginning of the round, O is the number of lines that are outaged during the round, I is the number of islands at the end of the round, and Y is the percentage of demand satisfied at the end of the round. We can see that the cascade of failures stabilizes in **c20** well before $r = 20$.



	No control				c20			
r	κ	O	I	Y	κ	O	I	Y
1	40.96	86	1	100	40.96	86	1	100
2	8.60	187	8	99	8.60	165	8	96
3	55.51	365	20	98	61.74	303	17	96
4	67.14	481	70	95	66.63	408	44	94
5	94.61	692	149	93	131.08	492	94	93
6	115.53	403	220	91	112.58	416	146	90
7	66.12	336	333	89	99.62	326	191	78
8	47.83	247	414	87	60.95	227	248	77
9	7.16	160	457	85	32.50	72	279	76
10	7.06	245	542	84	9.50	43	292	76
11	37.55	195	606	83	45.28	35	303	76
12	13.04	98	646	82	11.60	10	306	76
13	22.61	128	688	82	3.88	6	310	75
14	10.64	107	715	81	1.46	4	312	75
15	5.03	64	721	81	1.34	1	312	75
16	84.67	72	743	80	1.13	1	312	75
17	32.15	52	756	80	1.38	2	312	75
18	6.50	43	763	80	1.26	1	312	75
19	9.97	85	812	80	0.99	0	312	75
20	32.34	39	812	2	0.99	0	312	75

Figure 2: Cascade Evolutions [7]

Additionally, the author provides first-order methods to maximize the amount of demand satisfied at the end of the cascade, along with an algorithm that accounts for stochastic processes. For the sake of brevity, I will not describe them here.

3.3 Selective Node Removal Algorithm

Much of the research into attacks on complex networks is motivated by the observation that some nodes in a given network are more important than others with respect to load handling. The removal of these nodes has the capacity to cause the fragmentation of the network. Previous research has found that networks such as power grids or computer networks can easily fail due to the removal of one or several nodes since these removals can cause global cascades of overload failures. One strategy to reduce the likelihood of such a cascade is to intentionally remove nodes with small load and edges with large excess of load from the network. Though such intentional removals affect the network's ability to function as intended, the size of the cascade is reduced. Using the problem formulation described in [6], I will describe this selective node removal strategy.

3.3.1 Definition of the Network

We assume a network W where at each time step one packet is sent from node i to node j along the shortest path possible when i and j are in the same connected component. This packet may be divided into equal parts if there is more than one shortest path between i and j.

The load L_k on a given node k is defined as the total number of packets per unit of time that pass through k. If S_k is the connected component of k and $L_k^{(i,j)}$ is the contribution of the ordered pair of nodes (i, j) to L_k, then

$$L_k = \sum_{i,j} L_k^{(i,j)}. \tag{3}$$

Additionally, we define the capacity C_k of k to be

$$C_k = \lambda L_k(0) \qquad k = 1, 2, N, \tag{4}$$

where N is the initial number of nodes in W and $\lambda \geq 1$ is the tolerance parameter which guarantees that no node in the initial network configuration $W(0)$ is overloaded.

If D_{ij} is the shortest path length between nodes i and j and $\bar{D}_i$ is the average shortest path length from i to all other nodes in W, then the total load generated by a given node i is

$$L_i^g = \sum_j (D_{ij} + 1) = (\bar{D}_i + 1)(N - 1). \tag{5}$$

3.3.2 Strategies

There are four strategies that are part of the selective node removal algorithm, summarized as follows:

1. Remove the nodes with the smallest $\Delta_i \equiv L_i - L_i^g$ first.
2. Remove the nodes with the smallest closeness centrality $\bar{D}_i^{-1}$ first.
3. Remove the nodes with the smallest load L_i first.
4. Remove the nodes with the smallest degree κ_i first.

I describe these strategies in more detail below, along with justifying their usefulness in mitigating cascading failures.

Strategy 1: $\Delta_i \equiv L_i - L_i^g$
One strategy suggested in [6] as part of a selective node removal algorithm is to remove nodes with the smallest values for $L_i - L_i^g$ first. We define

$$\Delta_i \equiv L_i - L_i^g \tag{6}$$

to be the difference between the load on node i and the total load generated by node i. The most important nodes needed in order for the network to function are those which have a load L_i much greater than L_i^g, and so we do not want to remove these nodes in order to mitigate cascading failures. Likewise, nodes which have a load L_i much smaller than L_i^g generate more load than they handle, and so they are candidates for being removed first.

Strategy 2: Closeness Centrality
Another strategy suggested by [6] is to remove nodes with the smallest closeness centrality first. We define

$$\bar{D}_i^{-1} \tag{7}$$

to be the measure of closeness centrality for a given node i, recalling that

$$\bar{D} = \sum_i \bar{D}_i / N \tag{8}$$

is the average shortest path length between any two nodes and $\bar{D}_i$ is the average shortest path length from node i to all other nodes in W. Closeness can be considered as a means of determining how long it will take for load to spread from a node to other reachable nodes.

Strategy 3: Load
Yet another strategy suggested by [6] is to remove nodes with the smallest load L_i first. We have already defined load on a given node i to be the total number of packets passing through i per unit of time.

Strategy 4: Degree
The final strategy suggested by [6] is to remove nodes with the smallest degree κ_i first. Recall that the degree of a node is defined to be the number of links incident to it.

3.3.3 Algorithm Justification

In [6], only the first strategy that I have previously described is justified in detail. This is due to the fact that in random scale-free networks (SFNs), L_i^g and L_i are negatively correlated, and Δ_i, $\bar{D}_i^{-1}$, L_i, and κ_i are positively correlated. A scale-free network has a degree distribution that asymptotically follows a power law. Defining $P(k)$ to be the fraction of nodes in the network with k connections to other nodes, we have

$$P(k) \sim ck^{-\gamma} \tag{9}$$

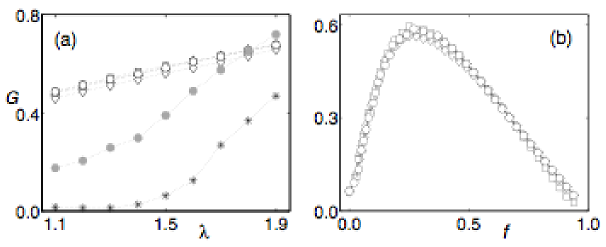


Figure 3: (a) Ratio G as a function of the tolerance parameter λ. Stars correspond to attacks without defense, while open circles, squares, triangles, and diamonds correspond to the intentional removals of nodes according to the strategies of defense (1) - (4), respectively.
(b) Ratio G as a function of the fraction f of nodes intentionally removed according to each of the strategies (1) - (4), for $\lambda = 1.5$. [6]

where c is a normalization constant and γ is a parameter with a value that is usually between 2 and 3. Power grids are generally considered to be SFNs, where power lines are edges and substations, generators, and transformers are nodes. So, the justification of one of the aforementioned strategies effectively justifies all four of them. I will not provide the proof described in [6]; however, figure 3 gives the ratio G as a function of the tolerance parameter λ and as a function of the fraction of the nodes intentionally removed. G is defined as $G = N'/N$, where N is the initial number of nodes in the largest connected component and N' is the final number of nodes in the largest connected component. As we can see from figure 3, selective node removal strategies can increase G by a factor of 6, potentially drastically reducing cascade propagation throughout a network.

4. FUTURE DIRECTIONS

4.1 Further Work on the Smart Grid GUI

The overall goal of the Smart Grid GUI is to help users visualize the network topology and cascade failure mitigation algorithms, along with making it easily extensible. As previously stated, I have made substantial progress toward creating the GUI itself, along with implementing the selective node removal algorithm. Figure 4 is a screenshot of what the application currently looks like with no network defined or algorithm selected. The application allows the user to add nodes, which the user can name and potentially add to a list of nodes that act as points of failure within the network (figure 5). I also include a screenshot (figure 6) without the node creation frame blocking the view of the main window. Finally, figure 7 is a screenshot of the edge creation frame that allows the user to name an edge, specify its endpoints, and select whether or not the edge will initially fail.

4.1.1 Refining the GUI

The Smart Grid GUI has the potential to be more than an environment to test cascade failure mitigation algorithms. In fact, by making the application more generalized, it could be used for a variety of networking simulations. However, in order to do this, I would need to better utilize JGraphT, which is a free Java class library that provides mathematical

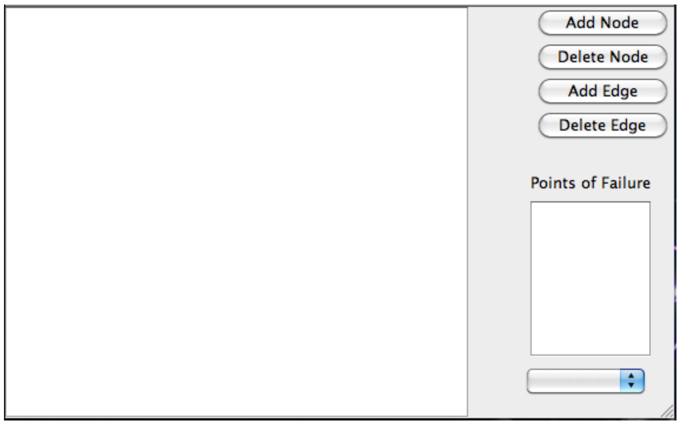
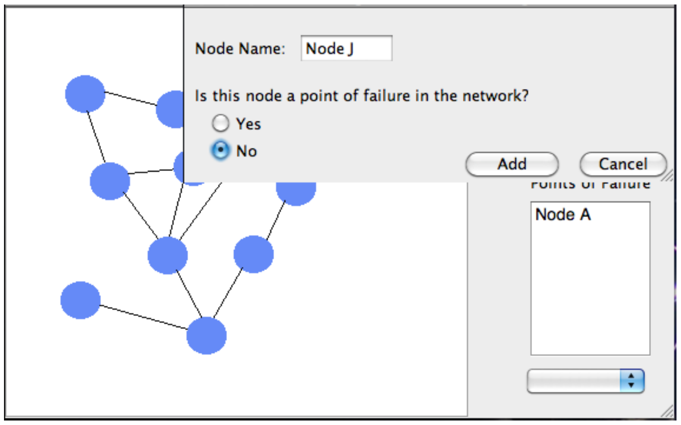


Figure 4: Smart Grid GUI without a network defined or algorithm selected.


Node Name: Node J

Is this node a point of failure in the network?

Yes [✓ No]

Figure 5: Smart Grid GUI with node creation frame.

graph-theory objects and algorithms. I initially began the implementation of node and link classes on my own; however, I decided to make subclasses for previously written graph classes in JGraphT. While these previously written graph, vertex, and edge classes do not describe all the attributes of a power grid, substation or line, they do provide an acceptable framework from which to derive more detailed classes. Additionally, JGraphT provides several well-known algorithms that are often needed by algorithms a user might wish to implement. For example, JGraphT provides classes that will find shortest paths, vertex covers, and cycle detectors.
In addition to better integrating the JGraphT library, future work on the Smart Grid GUI would provide more features for users. These features could include a statistics display, a tabbed environment to view different networks in the same window, and the ability to specify different link loads and failure times, among other things.

4.1.2 Further Algorithm Implementation

A more refined version of the Smart Grid GUI would have the ability to perform and visualize network simulations, along with aggregating data and providing valuable feedback for its users. The reason I chose to use Java was because concepts related to graph theory lend themselves easily to

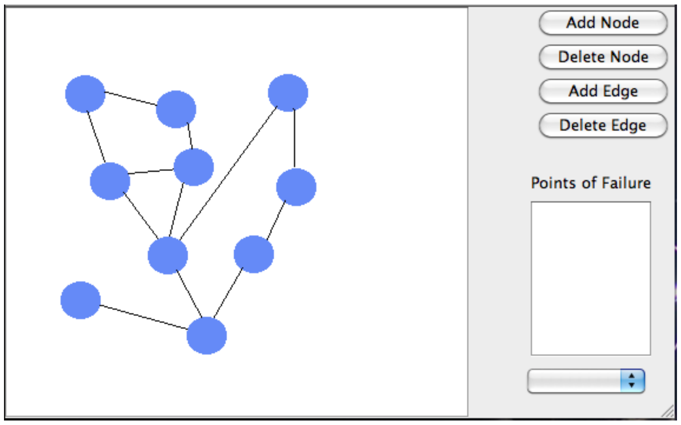
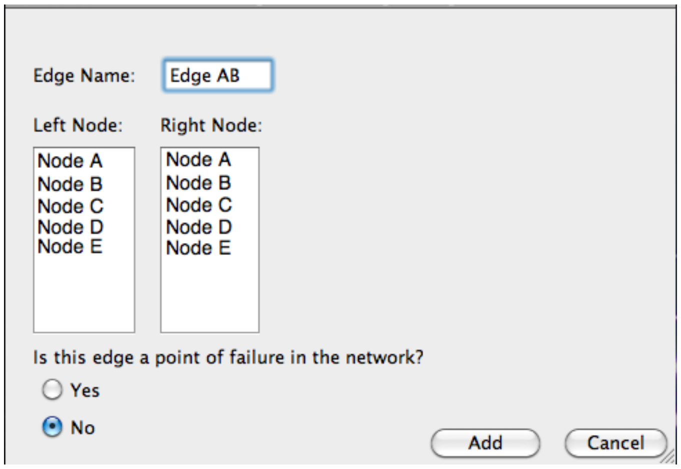


Figure 6: Smart Grid GUI with a network defined.



Edge Name: Edge AB

Left Node:	Right Node:
Node A	Node A
Node B	Node B
Node C	Node C
Node D	Node D
Node E	Node E

Is this edge a point of failure in the network?

Yes [✓ No]

Figure 7: Edge creation frame.

object-oriented programming. By creating a separate class for each algorithm implemented and having the user simply pass a network object to an instance of the class, one would ensure that everything remains cleanly organized and encapsulated. This would translate well to the GUI, which would allow the user to create a network and select procedures to perform using the provided buttons and menus.

5. CONCLUSIONS

Smart grid electricity systems have the potential to increase overall energy efficiency and reliability while decreasing costs for consumers. However, these systems can only be effectively implemented if the security vulnerabilities associated with them are eliminated. Modern power grid systems are especially prone to cascading failures, and in order for the smart grid system to be considered an improvement over the current power grid, this weakness needs to be fixed. By utilizing the data available at both the supplier and consumer ends of the grid, researchers can determine the best means possible to mitigate these types of failures.

However, there are many other security vulnerabilities that power grids face. By expanding the functionality of the Smart Grid GUI, users would be able to implement algorithms that handle different kinds of security issues, as opposed to just cascading failures. This would require some changes to the application I have designed. However, by maintaining a general framework that represents all the features of a power grid system, users would only need to add classes to specify cyber security and other algorithms.

6. ACKNOWLEDGMENTS

I would like to thank Steven Low, Kymie Tan, and Adam Wierman for their assistance and support.

7. REFERENCES

[1] Khurana, H., et al., *Smart-Grid Security Issues.* IEEE Security and Privacy, 2010. 8(1): p. 81-85.

[2] Prasanna, Srinivasa, et al., *Data Communication over the Smart Grid.* IEEE International Symposium on Power Line Communications and Its Applications, 2009.

[3] Zhenhua, J., et al., *A Vision of Smart Transmission Grids.* IEEE Power and Energy Society General Meeting, 2009.

[4] U.S. Department of Energy Office of Electricity Delivery and Energy Reliability, National Energy Technology Laboratory, *A Vision for the Modern Grid*, 2007.

[5] J.S. Simonoff, C.E. Restrepo, R. Zimmerman, *Risk Management and Risk Analysis-Based Decision Tools for Attacks on Electric Power*, Risk Analysis, Vol. 27, No. 3, 2007, pp. 547-570.

[6] A.E. Motter, *Cascade Control and Defense in Complex Networks*, Phys. Rev. Lett, 2004. 93(9).

[7] D. Bienstock, *Optimal Adaptive Control of Cascading Power Grid Failures*, 2010.