5-2022

Modeling Damage Spread, Assessment, and Recovery of Critical Systems

Justin Burns
University of Arkansas, Fayetteville

Modeling Damage Spread, Assessment, and Recovery of Critical Systems

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Justin Burns
University of Arkansas at Little Rock
Bachelor of Science in Computer Science, 2018

May 2022
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____
Brajendra N. Panda, Ph.D.
Thesis Chair

_____
Ukash Nakarmi, Ph.D.
Committee Member

_____
Lu Zhang, Ph.D.
Committee Member

ABSTRACT

Critical infrastructure systems have recently become more vulnerable to attacks on their data systems through internet connectivity. If an attacker is successful in breaching a system's defenses, it is imperative that operations are restored to the system as quickly as possible. This thesis focuses on damage assessment and recovery following an attack. A literature review is first conducted on work done in both database protection and critical infrastructure protection, then the thesis defines how damage affects the relationships between data and software. Then, the thesis proposes a model using a graph construction to show the cascading affects within a system after an attack. This thesis also presents an algorithm that uses the graph to compute an optimal recovery plan that prioritizes the most important damaged components first so that the vital modules of the system become functional as soon as possible. This allows for the most critical operations of a system to resume while recovery for less important components is still being performed. The thesis shows results from simulations using the recovery algorithm on data graphs with various parameters. After that, a second model is proposed that accounts for the time elapsed after an attack to perform a more precise damage assessment. By doing this, it can be determined how far damage can spread, then unaffected parts of the system can be released for possible use. Simulations are also done on this model to show the changes in damage assessment when different parameters are altered.

ACKNOWLEDGEMENTS

I would like to extend my deepest gratitude to my advisor and committee chair, Dr. Brajendra Panda. His guidance and encouragement have helped make my time in the Master's program and at the University of Arkansas wonderful experiences. I would also like to thank my committee members, Dr. Ukash Nakarmi and Dr. Lu Zhang, for being in my Master's committee. They were additionally of great help during my courses with them as instructors.

TABLE OF CONTENTS

LIST OF TABLES/FIGURES

1 INTRODUCTION

Critical infrastructure systems are those that are considered extremely critical to the functioning of a government or a country. As described in [2], critical infrastructures are like the vital organs of a human body that need to perform their own roles for the whole body to function efficiently and painlessly. The US Department of Homeland Security [3] declares that such systems are "so vital to the United States that their incapacity or destruction would have a debilitating impact on our physical or economic security or public health or safety." Therefore, the protection and smooth functioning of our nation's critical infrastructures are indispensable and cannot be ignored.

These systems are becoming prime targets of attackers – primarily state actors – and a major attack on one can cripple the economy of the victim nation. These systems are also more likely to be connected to the internet now to provide benefits like cost reduction (where large systems can be remotely managed over the public network), increased capability (by providing sufficient computing resources for infrastructure hardware with less capability power), and improved efficiency and transaction speed. This connectivity unfortunately makes it easier for attackers to hack into these systems. Consider the New York Times report about the attack on Colonial Pipeline [4]. While the details of the attack are not yet disclosed, a group of cybercriminals were able to compromise data systems using the internet, which resulted in Colonial Pipeline shutting down their pipeline. This outage affected mass transit and other industries across the entire U.S. East Coast and exposed a lack of preparation for such a crisis. This illustrates how an external system can have a relationship with a critical infrastructure system and how such relationships can be exploited to carry out an attack.

It is clear from past incidents and recent reports ([5]-[8]), to cite just a few) that attacks on critical infrastructures are occurring frequently, which indicates that prevention mechanisms are not enough to stop them. Thus, it is of utmost importance to aggressively prepare for post attack activities, which include damage assessment and recovery mechanisms that are critical to making the affected systems available at full functioning mode as soon as possible. This research aims at meeting this important goal.

This thesis proposes a framework that models damage spread within a set of data objects based on object dependencies and prioritizes making repairs to the most critical objects first. The framework is based on some of the models explored in critical infrastructure protection and uses a version of previously proposed repair methods that is modified to focus on meeting specific goals when determining the order in which repairs are made. Furthermore, a second model is proposed that assesses damage using the time elapsed since an initial attack to determine which section of a critical system is damaged with greater precision than usual. Experiments are performed to test both models by simulating critical systems with different parameters and comparing how damage assessment and recovery changes between the systems.

The rest of the thesis is organized as follows. Chapter 2 offers some work performed in this area and explains how this work is relevant to the goal of this thesis. Chapter 3 defines the problem that this thesis aims to build the two models for. In Chapter 4, the types of relationships that exist in data systems are presented and explained. Details on the damage recovery model are given in Chapter 5, which includes three subsections to establish the definitions, model description, and algorithm, and Chapter 6 presents results from the experiments done on this model. Chapter 7 covers the definitions, description, and algorithm for the damage assessment

model, and Chapter 8 shows the experiments and results from simulating that model. Chapter 9 concludes the work.

2 BACKGROUND AND RELATED WORK

This thesis aims to examine methods and frameworks used for database and critical infrastructure protection and apply it towards protecting a set of data objects. This section describes some of the publications that are relevant to the proposed framework. Various types of critical infrastructure objects are described in [17]. The focus of this thesis falls under network and network nodes, which are defined as a "structure with one dominating dimension", and junctions within the network, respectively. Furthermore, [18] establishes the concept of resilience, a cyclical process by which a system undergoes recovery, adaptation, and prevention between attacks. Turning this concept into a concrete value that can be used in risk assessment has been a point of interest in protecting systems [20]. Efforts have also been made to quantify vulnerability as a measurement that can be used to determine how frequently or severely a system can be at risk [19]. One of the major works on damage assessment and recovery within a database uses data dependency to find data affected by an attack to optimize recovery [9]. While this method relies on the direct relationships between data items, an alternate model to recover data from an attack instead uses the transaction log for assessment [10].

Kotzanikolaou et. al describe a model in [11] that assists in risk assessment for possible scenarios that can result in cascading failures within a CI system. For critical infrastructures with data-rich operations, the use of Cyber-Physical Systems can cause new vulnerabilities as described in [12]. Their model analyzes threats that can appear due to these vulnerabilities and analyzes the potential cascading damage they can cause. System dynamics modeling can also be

used to analyze disruptive events to characterize such disruptions to critical infrastructure by risk assessment and various impact factors as shown in [13].

Rehak et. al [14] model an infrastructure system as elements and linkages with different types of relationships establishing dependencies and interdependencies. They note that these elements can have varying criticality, causing some elements to cascade more damage into the system than others in the event of a failure. This work is important because by establishing criticality, they quantify damage within a system. This concept of criticality is used in later chapters to direct the optimal repair path of data objects.

This thesis also considers models that assist with recovery during an attack. In [15], an algorithm is proposed to restore damaged element paths by recursively breaking down demand flows into simpler problems. They use a centrality metric to rank damaged nodes and determine which ones should be repaired first and expand on the use of centrality to make repair decisions in further work [16]. This thesis uses the concept of centrality to rank data objects in a case where two or more are equally critical. In the proposed damage assessment algorithm, their method of simplifying damage paths is utilized to find the fastest route to restoring intermediate data objects. However, the novelty of the proposed approach is twofold: all components must be repaired within the system because data objects cannot have computations rerouted, unlike the network components in the work that has been reviewed, and the proposed method aims to restore the most important components first so that their functions can be restored while repairs to the system are still ongoing.

The work of Narayanan et. al [1] provides inspiration for the second model presented in this thesis. They present centrality metrics used to identify important water flows within a network of pipes using factors such as water volume and demand at different areas. While these

factors do not exist in a data critical system, there are similar ones that do exist, such as the frequency by which a node is updated with data from another node. If the elapsed time since an attack is known, this frequency can be used to trace damage spread along the nodes that can be affected by the damage. This is elaborated on in Chapter 7.

3 PROBLEM DEFINITIONS

In the occasion when an adversary information attack succeeds, the victim must have the capability to degrade gracefully and recover damaged data and/or services in real-time if it is to survive. It is necessary to immediately carry out damage assessment and recovery process in order to bring the systems to working states. Otherwise, the damage would spread to other unaffected systems that are interconnected. This happens when a valid user or an unaffected system module reads a damaged object during its computation and updates another object based on the compromised value, causing the latter damaged as well. As time goes on, more and more objects become affected in this manner causing the spread of damage to fan out through the system quickly.

For damage assessment and recovery purpose, information about all processes that have been executed must be stored in the log (more on this presented later). This will help in determining the relationships among the processes, thus helping in establishing the damage trail. Moreover, during recovery, the operations of processes that have spread the damage have to be undone and then redone in order to produce correct states of affected objects. The problems with existing systems are: (1) They do not store process execution information in the log, and they purge the log periodically, (2) Their recovery mechanisms are not designed to undo the effects of executed processes, (3) The size of the log, as it must not be purged, will make it almost impossible to continue the recovery process in real-time, and (4) During the damage assessment

and recovery process, the system remains unavailable to users. This delay induces a denial-of-service attack, which is highly undesirable in time-critical applications that the critical infrastructures are designed to provide. Due to massive amount of data in the log that needs to be processed, the problem becomes even worse.

The goal of this research is to develop fast, accurate, and efficient damage assessment and recovery techniques so that critical information systems not only survive the attacks gracefully but will continue to operate providing as many vital services and functions as possible even before the system is fully recovered. The next section explains how the relationships between different parts of critical systems can affect the spread of damage.

4 TYPES OF DAMAGE DEPENDENCIES

Attacks can affect not only data systems, but also software that produces data. This happens when the software is maliciously changed to perform unintended actions. This chapter covers possible attack scenarios involving damaged data objects and software and how they can cascade into other system.

4.1 Data to Data Damage

Data objects are frequently dependent on other data objects for computational updates. This makes their systems vulnerable to malicious attacks and cascading damage. Once a data object uses a damaged object to make a change, it becomes damaged too. This can happen to as many objects as the initially attacked objects have that are dependent on it. Figure 1 shows an example of cascading damage between data objects. If node C is targeted for an attack, then node E will become damaged due to its dependency on C. Then, node F and node G are also damaged because they are dependent on E.

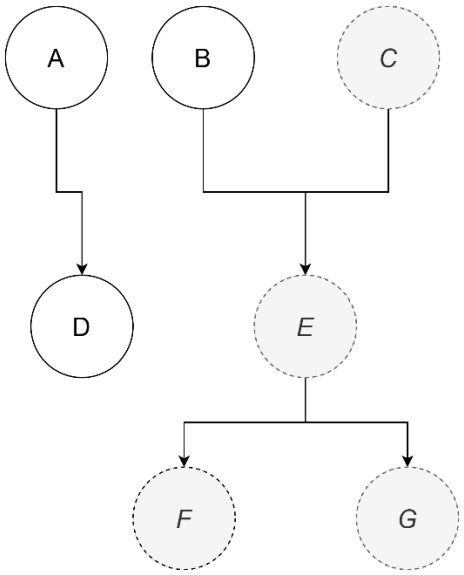

Figure 1. Cascading Damage from a Data Object

4.2 Software to Software Damage

If damaged software is used to influence computing done by another software, then the output of that software is also considered damaged. An example of a dependency between software is the use of libraries for applications. A damaged procedure within a library will result in damaged output for any software that calls that procedure. One difficulty with recovering from cascading damage from software to software is that it may be difficult to identify which cross-software calls used damaged procedures for computation. Consider a program P that uses two library functions A and B. A is an undamaged function and provides an undamaged output, while B is damaged and results in a damaged output. Even though it is known that the library is damaged, it is difficult to find which functions within the library are specifically damaged due to library already being compiled into binary format. Therefore, it is not known if A or B is damaged, and after the library has been recovered, P must be executed entirely to ensure that the correct output is obtained, which causes recovery time to increase. After recovery, the resulting output from A

is unchanged, while the output from B is corrected to the desired result. An example of software-to-software damage is shown in Figure 2. Even though the data is correct, a maliciously changed library can result in the software changing the output data.

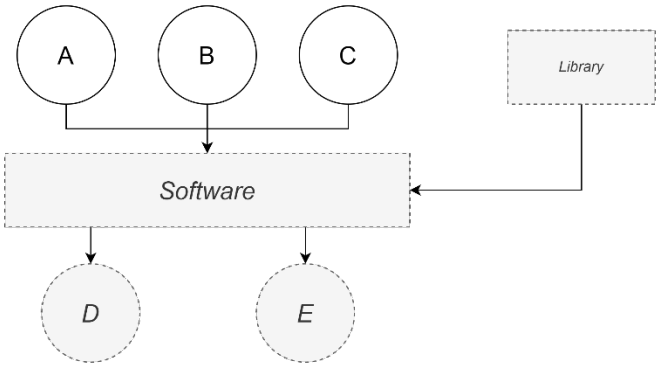

Figure 2. Cascading Damage through Software

4.3 Software to Data Damage

Any data changed by damaged software is also considered damaged. This is the simplest scenario involving damaged software because it is functionally the same as cascading damage between two data objects. After the software is repaired, any data that was damaged by the software must be computed again to complete recovery.

4.4 Data to Software Damage

Software that uses damaged data will not become damaged as a result. However, its output data will be considered damaged. This is because once a program is compiled, it cannot be changed by its input data. Therefore, if a software uses incorrect data to produce a damaged output, it is considered data-to-data damage instead. In Figure 3, a damaged node A is used as input for computation in a software. While the software itself is not damaged, its output nodes D and E become damaged.

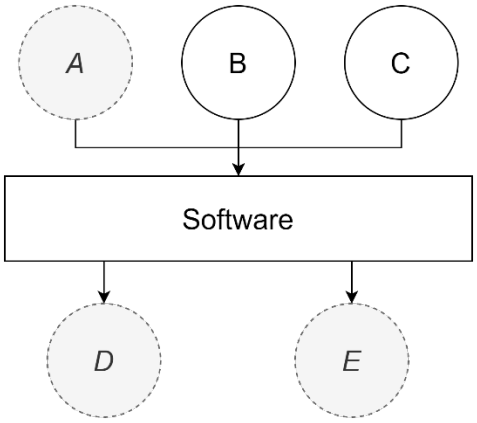

Figure 3. Cascading Damage from Data through Software

In the next section, the first model is presented. The aim of this model is to identify and recover damage optimally in detail. While it is important to consider the possibility of damaged software affecting a system, the model focuses on the spread of damage within data objects. This is because damage can spread much faster through data objects than in software, and usually does so more frequently. Furthermore, software can also be quickly fixed, while data objects are more numerous and may need many computations to get the desired output after repairs.

5 CRITICAL SYSTEMS DAMAGE RECOVERY MODEL

In this section, the first model is described in detail. The first subsection defines important graphs and metrics that are used for the first model. The second subsection provides details on how the model is built and is used to determine an optimal recovery plan. Finally, the algorithm used to implement the model is described.

5.1 Definitions

First, the concept of information flow in a system must be defined. This also establishes dependencies among various objects in the system and is used in the graph-based model.

Definition 1: *Given two objects O_i and O_j in a system, if the value of O_j is calculated using the value of O_i, there is information flow from O_i to O_j. Thus, O_j is said to be dependent on O_i and is denoted as $O_i \rightarrow O_j$.*

The above definition helps in determining the spread of damage in the system. That is, if an object is damaged, then all its dependent objects will be considered damaged. During recovery, the parent (pre-cursor) object must be recovered before any of its dependent objects can be recovered.

Next, the graph containing the set of objects and all possible paths among them is defined. This is called the Possible Paths Graph and it spans the entire system of objects and all dependency paths among them. An example of this graph is shown by Figure 4(a).

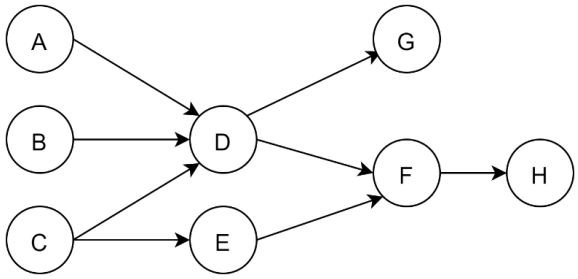

Figure 4a. The Possible Paths Graph (PPG)

Definition 2: *Consider a system containing the set of objects O. The Possible Paths Graph (PPG) is built by having a node N_i for each object in O. There exists an edge E_{ij} from N_i to N_j in the PPG if there is a possibility that information may flow from N_i to N_j, that is, N_j may be modified based on the value of N_i.*

The purpose of building a PPG is that it will help during the damage assessment preparation phase. By assuming the point of attack one can identify the set of items that may be affected consequently. Thus, security officers can be prepared for different types of eventualities.

The second set of objects contains the actual paths that were used to make changes in the system within a specified time span, which for the purposes of the third graph that will be defined, is usually the time passed since an object has been damaged. This set is represented by the Active Paths Graph (APG), and all objects and dependencies in this set exist in the PPG. This graph will help in determining the damage flow in case of an attack. Given an initial attack point (an object), one can determine which objects in the system may be affected by the attack and which ones will not be. Therefore, the ability of the system to carry out its intended functions can be calculated. That is, during the recovery process, the set of damaged objects will be made unavailable while the rest can be made accessible. Knowing which objects will remain unaffected, one will be able to identify what services the system will be able to offer while the recovery continues.

Definition 3: *The Active Paths Graph (APG) contains nodes N and edges E such that for every* $N_i \in N$ *and every* $E_{ij} \in E$, *both* N_i *and* E_{ij} *are also present in PPG, and* E_{ij} *illustrates an actual information flow; that is* N_j *was updated based on the value of* N_i.

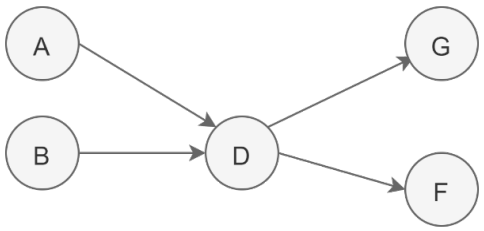

Figure 4b. The Active Paths Graph (APG)

Figure 4(b) provides an example of an Active Paths Graph and as can be seen it is a sub-graph of Figure 4(a). As discussed before, once an initial attack point is determined, the APG will help in accurately determining the damage flow and the set of objects affected by the attack. As discussed before, as time goes on, more and more objects will be affected as new objects will be updated based on the value of an affected object. Thus, to stop the spread of damage, all affected objects must be quickly identified and taken offline as soon as possible. This can be achieved by doing a flow assessment using the APG. This leads to the concept of actual damage spread path showing exactly which objects were affected by an attack. If a system is damaged, the spread of damage is represented as a third set of objects, the Damage Spread Graph (DSG). The set of objects and dependencies in this graph must exist within the APG, as damage spread occurs when objects make changes based on their dependencies. Like how the APG is a subsection of the PPG, the DSG is a subsection of the APG. Figure 4(c) is an example of what a damage path may look like. It is important to note that over time, a damaged object will always cascade its damage down to dependent nodes included in the APG. Definition 4 formally defines the DSG.

Definition 4: *A **Damage Spread Graph** (DSG) contains nodes N and edges E such that for every $N_i \epsilon N$ and every $E_{ij} \epsilon E$, both N_i and E_{ij} are also present in APG and every node in N is damaged through an attack on the system. Moreover, an edge E_{ij} depicts that N_i was damaged first and then N_j was damaged through the flow of information from N_i to N_j.*

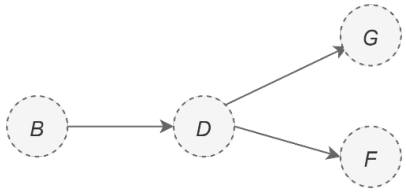

Figure 4c. The Damage Spread Graph (DSG)

Note that the edges between two objects may be bidirectional or recursive. For example, if an object O_j can have a dependency on object O_j and vice versa, then there will be a bidirectional edge between O_j and O_j. Similarly, if an object can be dependent on itself, it will result in a recursive graph. To clarify, let us consider an object "salary". When an employee receives an increment that is based on a percentage of the current salary of the employee, it causes the new salary to be dependent on the old salary and is depicted by using an edge from salary to salary itself. However, it must be noted that, for simplicity, neither bidirectional nor recursive edges are used in the APG or DSG. Rather, when an object is modified, it is noted as a new instance of the object, thus creating a new node for the object with the version number indicating when the object is in that state.

To minimize the time needed to restore the most important objects within a system of object dependencies, the following criteria used to determine the order in which repairs are made is defined:

Definition 5: *The criticality of a node N is its predetermined level of importance to the system's functions.*

This must be predetermined for the flexibility of the model to fit various systems and align the model with the goals of each specific system. For example, one system may need to prioritize certain components that other systems do not. The criticality of a component can be measured by various characteristics such as the intensity or scope of an impact caused by its failure as described in [14].

A positive whole number is assigned to each node N to represent criticality. A lower assigned value indicates higher criticality. For example, a node N_i with a criticality of 2 would be considered more important than a node N_j with a criticality of 4. It is important to note that

criticality values are not unique, meaning multiple nodes can have the same criticality value. When that happens, the following metric is used in the next definition to serve as a first "tiebreaker".

Definition 6: *Objects that have more damaged dependencies take longer to repair. Therefore, the repair time of a node N is defined as how many inward-flowing edges E^i it is receiving damage from.*

When two or more objects are assigned the same importance, the model chooses to first repair the one that has a lower repair time. For example, consider two nodes N_i and N_j that are equally critical. If N_i needs 5 other nodes repaired to repair it, and N_j needs 3 other nodes to repair it, then N_j will be repaired first, because its operation can be restored more quickly than that of N_i.

Definition 7: *The centrality of a node N is the number of outward-flowing edges E^o it has.*

The above metric is used to decide the next object to repair when two or more are equal in both criticality and repair time. An object with a higher number of E^o will have higher centrality. Figure 5a and Figure 5b show two subsections of a DSG that highlights centrality. As shown in Figure 5a, N_4 has three nodes that are dependent on it: N_1, N_2, and N_3, while as Figure 5b depicts, N_6 only has a single node N_5 dependent on it. Assume that the repair algorithm has repaired the parent node(s) of N_4 and that of N_6. To clarify the situation, N_4 and N_6 need not have the same parents; it is just that both are in line to be repaired next. In this scenario, repairing N_4 before N_6 reduces the repair time for the three dependent nodes of N_4 instead of only one of N_6, which can make future repairs be performed faster. Therefore, N_4 is considered to have a higher centrality than N_6.

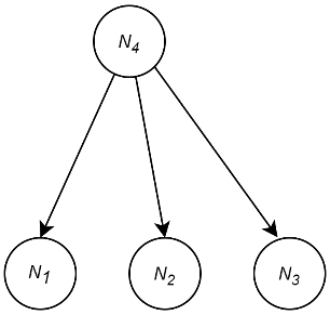

Figure 5a. A parent node with high centrality

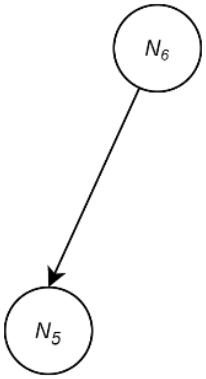

Figure 5b. A parent node with low centrality

Definition 8: *The relationships between all nodes n in a data system can be expressed by an adjacency matrix A, where each element represents an edge $e_{i,j}$ between nodes n_i and n_j such that the parent node is given by the element's row and the child node is given by the element's column.*

The value of each element is binary – if an edge going from one node to another exists, then the value of its respective element is 1, otherwise, it is 0. Therefore, for each element $a_{i,j}$ in A:

$$a_{i,j} = \begin{cases} 1 \text{ if } (i,j) \in N_i \\ 0 \text{ if } (i,j) \notin N_i \end{cases}. \qquad (1)$$

Figure 6 depicts the PPG shown in Figure 4a. as an adjacency matrix. Each cell with a value of 1 is an existing edge on the graph. For example, node D has 3 parent nodes and 2 child nodes. The edges coming from nodes A, B, and C are highlighted on D's row. Likewise, its outgoing edges toward nodes F and G are highlighted on D's column. All node relationships are translated this way, so as there are 8 edges in the graph, there is also 8 highlighted elements in Figure 6.

Since an adjacency matrix can be used to represent the relationships between data nodes in a uniform manner, it is used for the implementation of the damage recovery planning algorithm, which will be covered in the next two subsections.

	A	B	C	D	E	F	G	H
A	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0
D	1	1	1	0	0	0	0	0
E	0	0	1	0	0	0	0	0
F	0	0	0	1	1	0	0	0
G	0	0	0	1	0	0	0	0
H	0	0	0	0	0	1	0	0

Figure 6. An adjacency matrix of Figure 4a.

5.2 Model Description

The model uses the three graphs defined in the previous section to construct a representation of a given system and its sustained damage from the time of the initial attack. The PPG is a preprocessed map of all components and dependency paths within a system. It is assumed that it

is known how much time has passed since the initial attack and build the APG by including components and dependency paths that were used in a transaction log in that period. By knowing the component where the initial attack occurred, the DSG is built by tracing the damage through the transaction log. For damage to spread from one component to the next, it must follow two criteria: 1) there is a damaged node N_i that has an edge E_{ij} flowing from it to node N_j and 2) E_{ij} is used for a transaction while N_i is damaged. For the DSG to exist, the initial attack must occur within the APG, otherwise there is no cascading damage.

The goal of the model is to find the optimal sequence of repairs to restore the most important operations of a system as quickly as possible. The metrics defined in the previous section are used to decide which components should be repaired first. The first metric is criticality – the most critical components must be restored first to resume important operations. However, these components may also be dependent on other components that are damaged. These components must be repaired first before the base component can be repaired. At this point, the same problem is applied to the dependency components, and the most critical one is chosen first. If there is a tie, then components with a lower repair time are picked first. For example, a component that has two damaged parent components will be prioritized over a component with three or more damaged parent components if both components are equally critical.

To clarify, consider the graph presented in Figure 7. As shown in the figure, nodes N_1, N_2, and N_3 are dependent on N_4. Assume that the damage assessment method identified N_4 as damaged; thus, nodes N_1, N_2, and N_3 are also identified as damaged. During the recovery process, N_4 was recovered before the other three nodes. However, since it has three dependents all of which are damaged, the question is, which one should be repaired first. As the goal is to

have the vital functions of the system to be made available before the other operations, the algorithm would choose the node among N_1, N_2, and N_3 having the most criticality.

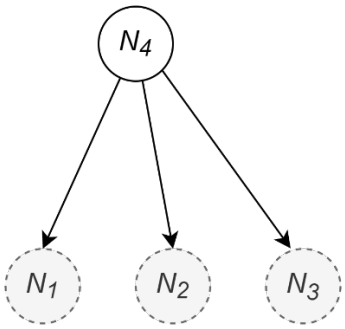

Figure 7. Recovery Sequence Decision

Repair time is not affected by how the parent components are ordered. For example, consider two scenarios with nodes N_1, N_2, N_3, and N_4. In the first scenario, node N_1 is dependent on nodes N_2, N_3, and N_4. In the second scenario, node N_1 is dependent on node N_2, node N_2 is dependent on node N_3, and node $N3$ is dependent on node N_4. In both scenarios, nodes N_4, N_3, and N_2 must all be repaired before node N_1 can be repaired, so N_1 will always have the same repair time. In short, repair time can simply be considered as the number of upstream components. Similarly, the third metric, centrality, can be considered as the number of downstream components. This is the third metric used to determine repair order in case multiple components are equal in criticality and repair time. This metric is not prioritized over the first two because it does not directly contribute toward the stated goals of the model, but it can optimize future repairs by lowering the repair time of more components than repairing other components would.

5.3 Model Algorithm

First, the primary objective of the model is described. Consider the notations used in the following table:

TABLE I. NOTATIONS

Notations	Descriptions
$P = (V, E)$	Possible Path Graph
$A = (V_A, E_A)$	Active Path Graph ($V_A \subseteq V, E_A \subseteq E$)
$D = (V_D, E_D)$	Damage Spread Graph ($V_D \subseteq V_A, E_D \subseteq E_A$)
$D = (V_C, E_C)$	Critical Node Graph ($V_C \subseteq V_D, E_C \subseteq E_D$)
δ_{ij}	Decision to fix edge i to j
δ_i	Decision to fix node i
t_i	Time to fix node i
c_i	Centrality of node i
P_{ij}	Dependency indicator of node i and j

The objective is to find $\min \sum_{i \in V_D} t_i \delta_i$ subject to

$$\delta_i \sum_{j \in V_C} P_{ij} \leq \sum_{j \in V_C} P_{ij} \delta_j \quad \forall i, j \in V_C \qquad (2)$$

$$\delta_i c_i \geq \sum_{(i,j) \in E_C} \delta_{ij} \quad \forall i \in V_C \qquad (3)$$

$$P_{ij} \in \{0,1\} \quad \forall i, j \in V_C \qquad (4)$$

$$\delta_i, \delta_{ij} \in \{0,1\} \quad \forall i \in V_C, (i,j) \in E_C \qquad (5)$$

That is, the goal is to minimize the time required to fix all critical nodes subjected to conditional constraints of the system. To make sure that each preceding nodes of i are fixed before node i being processed, condition (2) is used. For example, if there is a node j connecting to i but in a

prequel order, the sum product of all nodes j status and dependency indicator P_{ij} should be greater or equal than the product of sum of all dependency indicator P_{ij} with node i. To make sure that there would not be more out-going flows than the given capability of node i, equation (3) is imposed to make sure the total out-going edge would not surpass the centrality of node i. Conditions (4) and (5) were built to impose the binary attribute of the dependency indicator P_{ij}, the decision whether to fix node i or edge from node i to node j.

The algorithms provided in this section use the model described in the previous section to compute the optimal order of repairs to restore the most important functions of a system first. When an attack occurs, an Intrusion Detection System (IDS) is expected to identify the attack and provide the initial point of damage. The working principles of IDSs are not within the scope of this work and so, not described here.

Algorithm 1 creates the PPG from the system's data. The PPG must record all possible dependency paths, including those that may not have been used by the system yet. Therefore, it is necessary to consult the system's designers so the algorithm can be provided full information on the system's data objects to fully construct the PPG. As stated in Definition 8, the PPG, APG, and DSG are all represented by a binary adjacency matrix, where child nodes and parent nodes are represented by the columns and rows, respectively. Each element in one such matrix is the edge between the given parent and child node, with a value of 1 indicating that the edge does exist in the graph and a value of 0 indicating that it does not exist.

Algorithm 1: Initializing the Possible Paths Graph
Result: Adjacency matrix M^{PPG} representing the PPG
1 for each object O_x and O_y in the system
1.1 if edge E_{xy}can exist
1.2 $M^{\mathrm{PPG}}(x, y) = 1$
2 Return M^{PPG}

After receiving notification from an IDS, a precise damage assessment is performed. If the damage assessment process is unable to make accurate assessment, i.e., in case a damaged node is not correctly identified, it and its dependent nodes, which are also damaged, will remain unrecovered. This will result in valid users or procedures reading them and spreading damage by updating other objects, as discussed earlier. For a detailed discussion on damage assessment, one may review [9] and [10], which were developed particularly for database systems. However, the methods are still applicable to critical infrastructure systems. A basic mechanism to carry out the assessment is provided below.

Damage assessment begins with the APG, which shows the actual dependency relationships among the objects in the system (Note that the APG can be built as transactions are executed and dependencies are established among various nodes of the PPG). Given the initial attack point, the corresponding node is then marked as damaged. This is the starting node of the DSG. Then by scanning the log from the corresponding location of the attack point, transactions that read the marked node are identified. Any objects written by those transactions are then marked as damaged in the APG. This process continues until the end of the log. Finally, all unmarked nodes and the edges showing their dependencies are removed. The resulting graph is the completed DSG.

The APG is constructed in Algorithm 2. This algorithm reads the transaction log and creates a node each time a new data object is mentioned in the log. The object's dependencies are depicted as the node's edges in the APG. When an existing data object is updated, it becomes a new object in the transaction log. As described earlier, this is done to prevent recursive dependencies in the APG.

Once damage assessment is carried out, recovery procedure must begin immediately in order to make the system operational quickly. Algorithm 3 is used as the main procedure to initialize an

object set for repairs. The algorithm starts by initializing the set of damaged objects O. Each node N within O consists of a system component and its relationships with other nodes in O. As mentioned previously under Definition 4, some system components may have recursive or bidirectional dependencies between each other. Therefore, system components can have repeat nodes within O to represent their different versions. Each node is assigned values for criticality, repair time, and centrality. Using those metrics, the algorithm determines an initial target node N_0 based on criticality. If there are two or more nodes with the highest criticality, then the node with the lower repair time is selected. In the event of another tie, the node with higher centrality is selected. Further ties are broken by random selection. N_0, along with O and the repair queue Q, are used to make the first call to the recursive function Algorithm 3.1 at step 4.5. Algorithm 3 proceeds until O is completely empty, and then the repair queue is finalized, and Q is printed.

As previously discussed, a node must have its parent nodes repaired before it can be considered eligible for repairs. Algorithm 3.1 ensures that nodes are scheduled for repairs in the proper order while still adhering to the rules set for determining priority. It does this by using a while loop to check the currently selected node N for repair eligibility. If N is eligible for repairs, then it is removed from O and Q is updated, then returned. If N is not eligible, then O', a subsection of O made up of all dependency paths above the currently selected node is created and used to find the next highest priority node N' within O'. Algorithm 3.1 is recursively called using N' and O', which can either result in the node's repair or another node being selected for repair again. The recursive nature of this algorithm ensures that each time a decision needs to be made on which node needs to be repaired next, it will prioritize criticality and efficiency among all the nodes that can be repaired at any given step. In this way, the bulk of the work done by the algorithm is choosing the next object for repair within each iteration. Each function call will result in one object being

repaired and $n-1$ additional function calls, where n is the number of nodes within the set of nodes being passed. Since repaired objects need to be removed from the DSG, function calls will need to update and return the global DSG and Q.

Algorithm 2: Initializing the Actual Paths Graph
Result: Adjacency matrix M^{APG} representing the APG
Parameters: transaction log T, containing the set of edges E that were used to make updates
1 For each edge E_{xy} in T
    1.1 $M^{\mathrm{APG}}(x,y)=1$
2 Return M^{APG}

Algorithm 3: Initialization for object set repair
Result: Queue of objects ordered by repair priority
1 Initialize set of damaged objects O
2 Preprocess object priority using criticality, repair time, and centrality
3 Initialize repair queue Q
4 while *O has damaged nodes remaining*
  4.1 Select the highest critical node(s) N within O
  4.2 if *Two or more nodes are tied for highest criticality*
    4.2.1 Select the node(s) N with the lowest repair time R within O
  4.3 if *Two or more nodes are tied for lowest repair time*
    4.3.1 Select the node(s) N with the highest centrality within O
  4.4 if *Two or mode nodes are tied for highest centrality*
    4.4.1 Select a single node at random from those still tied
  4.5 Update repair queue(N_0, O, Q) $\rightarrow Q$
5 Print Q

Algorithm 3.1: Recursive repair function
Result: Schedules a node N for repairs and returns the updated repair queue Q
1 Update repair queue(*Selected node N, object set O, repair queue Q)***:**
2 while *Current object has unrepaired dependencies*:
  2.1 Create subset of damaged nodes O' of all nodes N' and edges E' that N is dependent on
  2.2 Select the highest critical node(s) N' within O'
  2.3 if *Two or more nodes are tied for highest criticality*

2.3.1 Select the node(s) N' with the lowest repair time R within O
2.4 if *Two or more nodes are tied for lowest repair time*
2.4.1 Select the node(s) N' with the highest centrality within O
2.5 if *Two or mode nodes are tied for highest centrality*
2.5.1 Select a single node at random from those still tied
2.6 Update repair queue(N'_0, O', Q) → Q
2.7 Remove the most recent object in repair queue from O
3 Repair N
4 Add N to Q
5 Return Q

The algorithm produces a list of system nodes in the order in which they should be repaired. Recovery procedure then continues to the next step to begin repairs on the system. It is important to note that while repairs are simulated by the algorithm, the process for repairing the actual components of the system is not within the scope of this work.

6 EXPERIMENTS AND ANALYSIS FOR RECOVERY MODEL

6.1 Experiment Description

The efficiency of the recovery model was evaluated by using it to find the number of nodes needed to repair every critical node in a DSG simulation with various parameters. The simulation constructed a DSG with randomly assigned relationships between nodes. The parameters used to build each DSG were total nodes, percentage of critical nodes, maximum number of parent nodes per node, and maximum number of children nodes per node. After the DSG was built, the set of critical nodes was randomly picked from the entire DSG for the model to compute the required repairs. For each simulation, 25 different sets of critical nodes were tested, and the average

number of required repairs was reported by the simulation. Four experiments were run to test the changes to the number of required repairs caused by changing each parameter. For every experiment, the control variables were set to be 10,000 total nodes, 5% critical nodes, and a maximum of 6 parent nodes and 6 child nodes per node. Then, this method was compared to modified versions of two common traversal algorithms: breadth-first search (BFS) and depth-first search (DFS). The BFS algorithm repaired nodes starting with all the root nodes, then all the children of the root nodes, and then the next set of child nodes until it reached the bottom. On the other hand, DFS repairs a single parent node, then repairs one of its child nodes, and repeats that process until it reaches a node with no children. When that happens or if it has already repaired all child nodes for a given node, it goes back up to the previous node and repairs another child node. The exception to this process is if a node has other parent nodes that have not been repaired yet. The algorithm will break the traditional BFS or DFS order to ensure that the node is eligible for repair. The results for the BFS and DFS repair algorithms are included with the model's results.

6.2 Results and Analysis

Two sets of graphs are presented for each round of simulations in Figures 8 and 9. Figure 8 consists of four bar graphs highlighting the different in required repairs between the model's algorithm and BFS and DFS traversals. Figure 9 has four graphs showing the changes in the percentage of nodes repaired between each parameter of a simulation. The results for required repairs based on total nodes are shown in Figure 8a. Figure 9a. shows that the required repairs maintain a constant ratio between it and the total nodes with the given control variables, as each experiment resulted in a little less than 25% of the total nodes needing to be repaired.

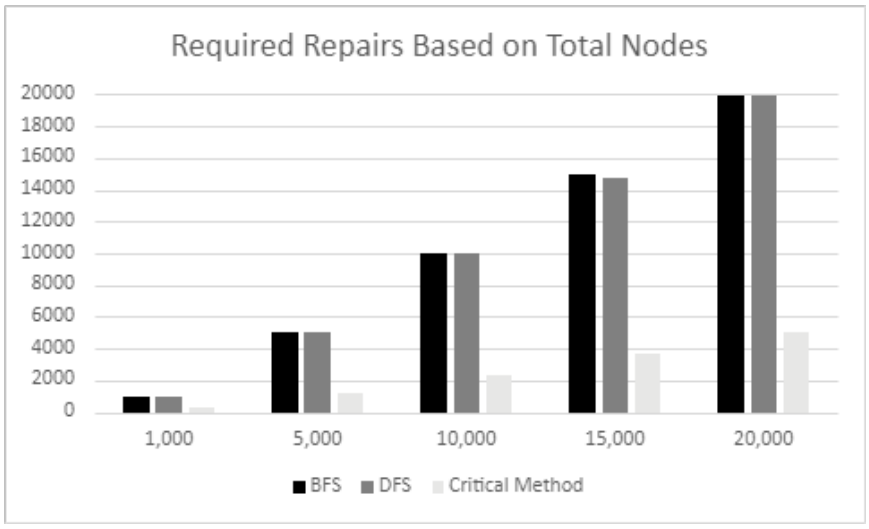

Figure 8a. Required repairs based on total nodes

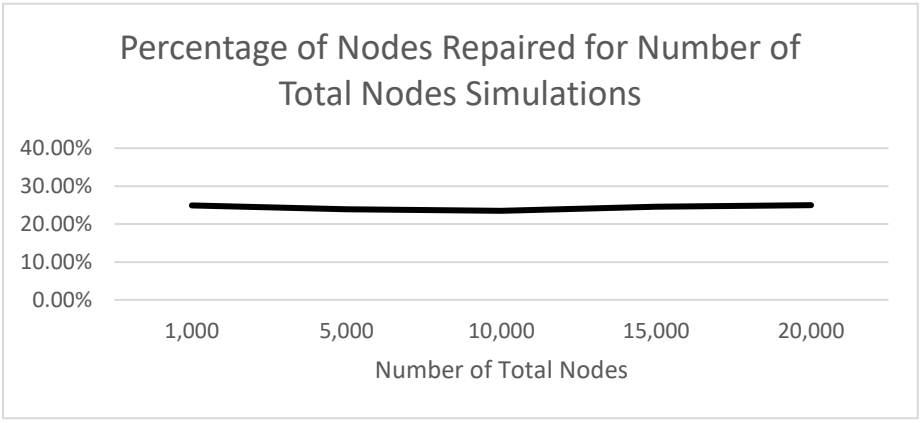

Figure 9a. Changes in required repairs based on maximum number of total nodes

Figure 8b. shows the results for required repairs needed for different percentages of critical nodes. While the number of required repairs increases as the percentage of critical nodes increases, less additional repairs are needed when the percentage is higher. Therefore, a high number of critical nodes require less repairs per critical node than a lower number of critical nodes. This is affirmed in Figure 9b., as percentage of nodes repaired steadily rises as the percentage of critical nodes increases.

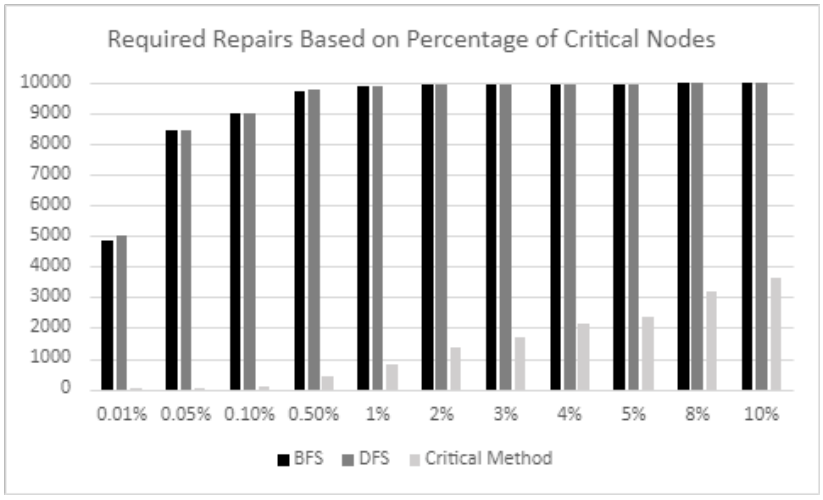

Figure 8b. Required repairs based on critical node percentage

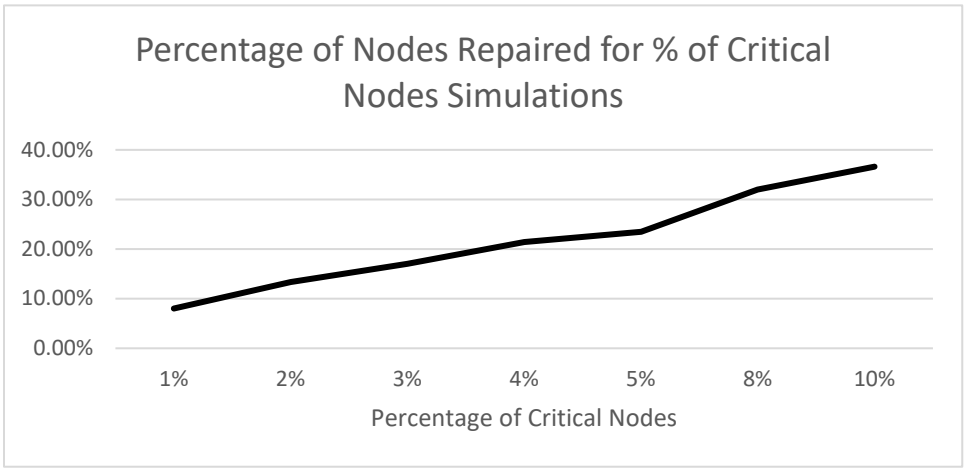

Figure 9b. Changes in required repairs based on percentage of critical nodes

Figures 8c. and 8d. show the change in required repairs for different maximums of children and parent nodes per node respectively. With a low number of maximum children per node, more repairs are required. This is because the system of nodes becomes narrower in shape. In contrast, when there is a higher maximum of parent nodes per node, the required repairs see an increase. This indicates that a critical node is more likely to have additional parent nodes, which would require more repairs than a critical node with fewer parent nodes. These changes mirror each other

in Figures 9c. and 9d. as well. However, when the maximum number of child or parent nodes becomes high enough, it appears to have less of an effect on the number of nodes repaired as it does at lower values.

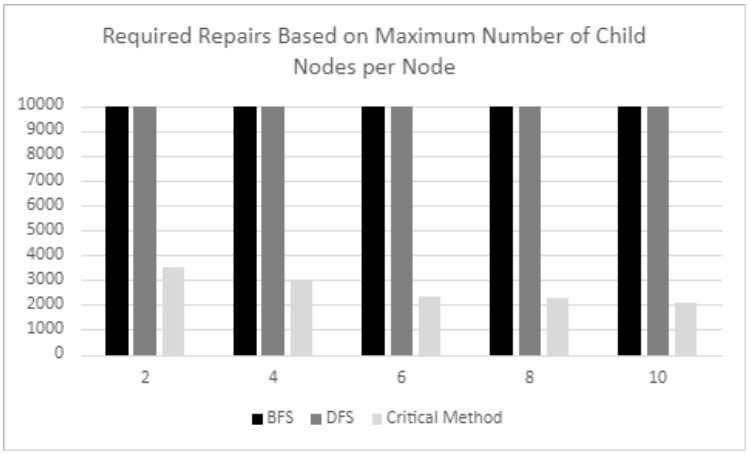

Figure 8c. Required repairs based on maximum number of child nodes

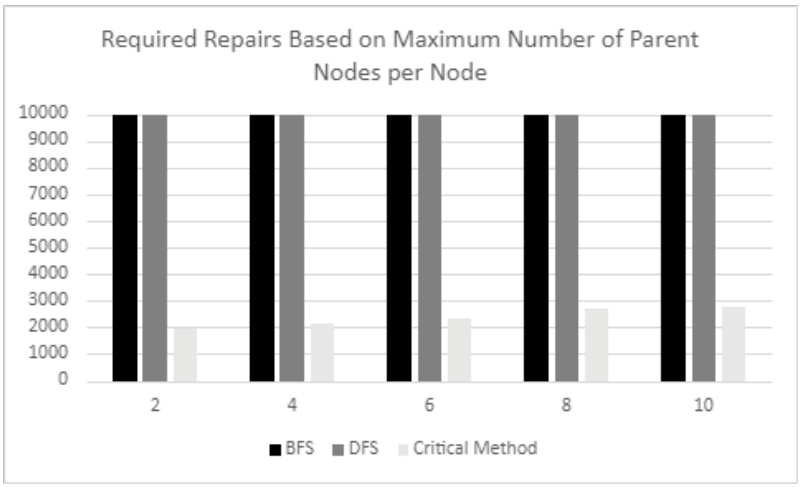

Figure 8d. Required repairs based on maximum number of parent nodes

Across all graphs, targeting specific nodes for repair drastically shortens the time needed to recover critical functions, as the average required repairs for the BFS and DFS algorithm in most simulations was slightly less than the total number of nodes in the simulation.

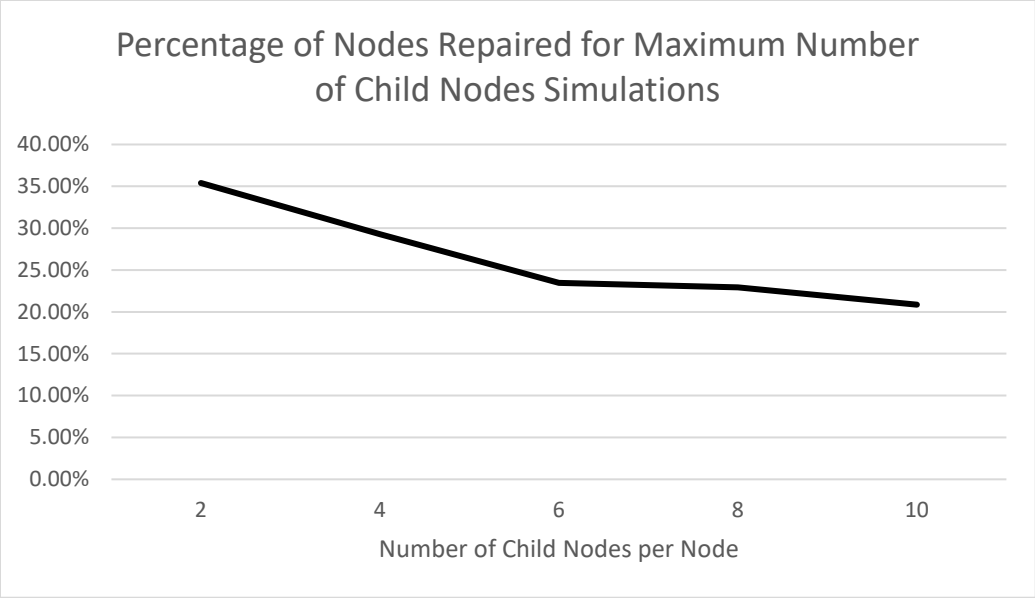

Figure 9c. Changes in required repairs based on the maximum number of child nodes per node

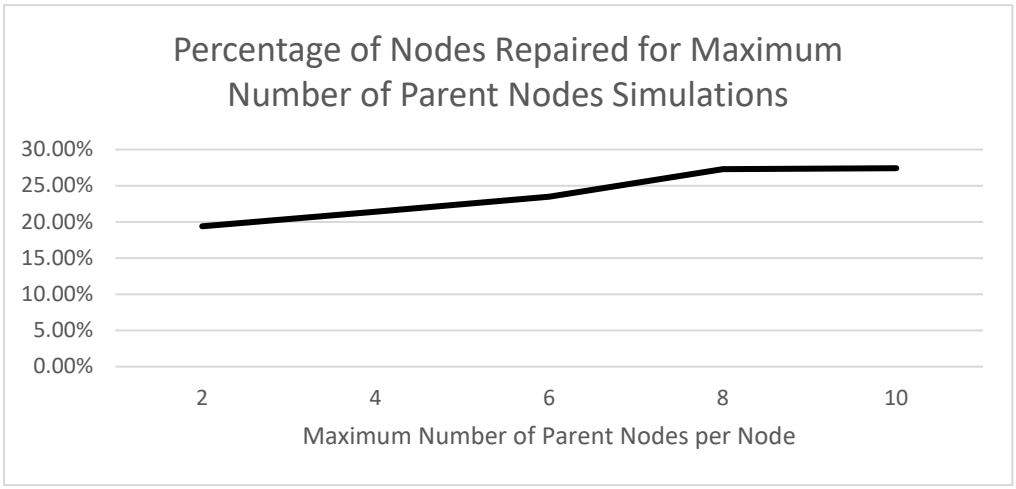

Figure 9d: Changes in required repairs based on the maximum number of parent nodes per node

6.3 On BFS and DFS Algorithms

Before running the simulations, the model's algorithm was expected to outperform BFS and DFS searches due to it specifically targeting critical nodes for repair first. However, these two

algorithms ended up traversing nearly the entire system before repairing all critical nodes. The best way to explain this is to consider the order by which each algorithm repairs nodes as a static ordered list. The model's algorithm sorts this list after nodes are assigned criticality, so the final critical node that needs to be repaired ends up being close to the front of this list. On the other hand, BFS and DFS do not consider criticality when sorting this list, so each node's criticality on their lists after sorting is effectively random. What determines how many nodes need to be repaired is ultimately the final critical node in the list, so the odds of this node not being in the final thousand or even hundred nodes are quite low. Therefore, BFS and DFS will need to repair almost all the nodes except for when the percentage of critical nodes is lower. When that happens, it is more likely that the final critical node will be closer to the middle of the repair order instead.

7 TIME-BASED DAMAGE ASSESSMENT MODEL

This chapter describes the second proposed model, the time-based damage assessment model. Like the previous model, this chapter has three sub-chapters: first, any necessary definitions are established. Next, how the model functions and why it is important is described. Finally, the algorithm for the model is presented.

7.1 Definitions

Recall that the previous model used an adjacency matrix to represent a data system. In this model, the adjacency matrix is once again used, however, a second matrix is defined to further establish the relationships between nodes in the system.

Definition 9: *The time-frequency matrix T is an extension of the adjacency matrix. Its rows and columns are the same as the adjacency matrix. However, the value of an element in the time-*

frequency matrix is the maximum amount of time that can pass between each instance a parent node updates a child node.

Unlike the adjacency matrix, an element can have a value in this matrix even if there is not a direct edge between the two nodes. This can happen if there is at least one existing path within the data system where the child node can indirectly depend on data from the parent node. For an indirect connection, the value of this element is the sum of all the direct connections along the "shortest" path between the two nodes.

$$t_{i,j} = \begin{cases} f_j^t \text{ if } a_{i,j} = 1 \\ min \sum_{k=1}^{E_k} f_y^t \text{ if } a_{i,j} = 0 \end{cases} \qquad (6)$$

It is important to note that the frequency of an edge is not always the same for every update. Therefore, three time-frequency matrices with the following values are established for the model's use: the maximum frequency of each edge, the minimum frequency of each edge, and the frequency of the most recent update of each edge. How these are used is explained in the following sub-chapter.

7.2 Model Description

The goal of this model is to assess the maximum progress that damage within a system can make given the amount of time passed since an attack on the initial node. Figure 10 visualizes the steps taken by the algorithm to make such an assessment. Consider a scenario where node B is damaged by an attack, and 160 seconds have passed since the attack. All the paths that are dependent on node B must be traversed to find out if they have been damaged. Each edge has a frequency range by which the number of seconds that pass before the next update is chosen from with a specified minimum and maximum value. Since it is of the upmost importance that the

spread of damage is never underestimated, the minimum value of this range is used to estimate the damage. In this scenario, the minimum frequency edge between B and D is 10 seconds. D is then assumed to be damaged, and 10 seconds are subtracted from the total time for determining if D has any dependent nodes that can be damaged. The edge from D to G has a minimum frequency of 180 seconds, which is greater than the 150 seconds that could have passed since D was damaged, so G is assumed to be safe for now. However, the minimum frequency of the edge from B to E shows that E can be damaged with 40 seconds to spare, and the edge from E to G has a minimum frequency of only 30 seconds. Therefore, G is now considered damaged through E, even though it could not have been damaged by D. This example shows the need for an algorithm performing this task to be thorough and that the worst-case scenario must always be accounted for during damage assessment.

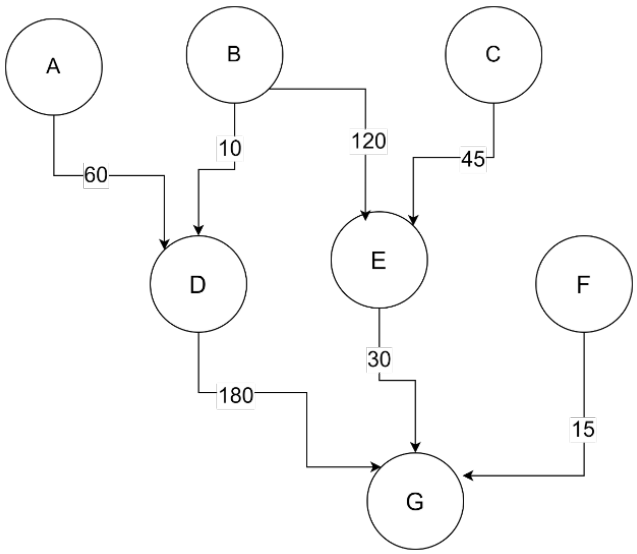

Figure 10a. Initial frequency graph

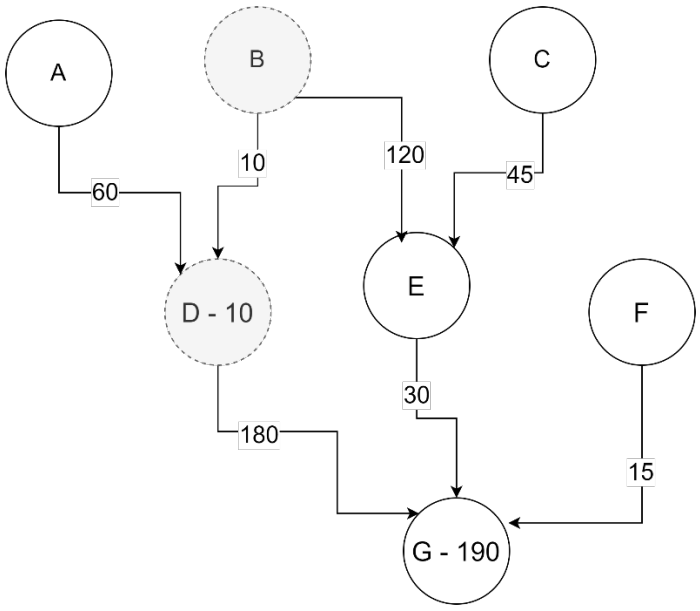

Figure 10b. Updated graph after estimating one path

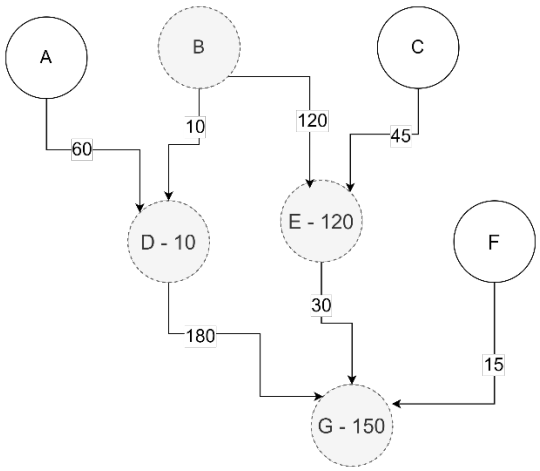

Figure 10c. Updated graph after estimating two paths

7.3 Model Algorithm

This sub-chapter describes how the algorithm for the model is implemented. Algorithm 4 shows how the model traverses a data system to determine damage spread. Depending on the type of time-frequency matrix used, the algorithm can estimate the worst-case or best-case damage

spread by using the minimum or maximum frequencies of edges respectively or compute the actual damage using a time-frequency matrix that has the frequency values for the most recent updates used on each edge. While the actual frequency values are not assumed to be known, this is used when performing simulations to compare the estimated damage with the actual damage.

Algorithm 4
Result: Finding the fastest paths for damage spread
Parameters: node graph *n,* adjacency matrix *e*, time-frequency matrix *t*

1 Initialize node graph where n' is the first damaged node and n'' time since the attack
2 Set current node n_c to n'
3 While n_c has children nodes that have not been traversed
    3.1 Pick a child node n_y to traverse
    3.2 Subtract $t_{c,y}$ from n_c^t for n_y^t
    3.3 if new $n_y^t >$ old n_y^t (including the initialization value of 0), update with new value
    3.4 set n_c to n_y and mark $e_{c,y}$ as traversed
    3.5 While n_c does not have a child node that has not been traversed and n_c has a parent node that has been traversed
        3.5.1 set n_c to previous node

8 EXPERIMENTS AND ANALYSIS OF THE DAMAGE ASSESSMENT MODEL

In this chapter, an attack on a data system is simulated using the adjacency matrix, then the algorithm is used to estimate the damage caused by the attack. The estimated damaged is compared to the actual damage simulated, and then the entire subtree of the attacked node is counted for further analysis.

8.1 Experiment Description

First, the adjacency matrix is built given a specified number of nodes. Edges between nodes are randomly created until the required parameters are satisfied. Then, each edge is assigned a random maximum frequency, minimum frequency, and an actual frequency chosen between the

former two values. The next step is to simulate an attack. Since the random generation of the graph is volatile and may result in insufficient data if the initially attacked node is also chosen at random, the node with the largest subtree within the graph is deliberately chosen instead. While this may not always be the case for real scenarios, the generated graph can instead be envisioned as a subgraph of a larger, unseen network. The actual damage is simulated using Algorithm 4 with the actual frequencies of the edges as the time-frequency matrix parameter. Finally, damage assessment is performed using the method. The only information that is assumed to be known is the initially damaged node, the amount of time since the attack, and the minimum frequency of each edge. The estimated damage is then returned and compared with the actual damage and how many nodes are in the subtree.

For this experiment, six different variables were tested for their influence on how many nodes are estimated to be damaged, how many nodes are actually damaged, and how big the difference between those two numbers are. Those variables are the total amount of nodes generated, the total time elapsed since the initial attack, the minimum frequency an edge can have, the maximum frequency an edge can have, the maximum number of child nodes a node can have, and the maximum number of parent nodes a node can have. The default values used for each variable are 3000 total nodes, 300 seconds of total time, a minimum frequency of 30 seconds, a maximum frequency of 60 seconds, and 6 nodes for both the maximum number of child nodes and parent nodes. The same results from the simulation using the default values are included for all the experiments, and each experiment's variable is simulated again with the value being incremented and decremented twice, for a total of 5 simulations per experiment. The results are provided in the next sub-chapter.

8.2 Results and Analysis

The results for each variable are presented in the following graphs. Figure 11a. shows the change in damage assessment when the total nodes in the graph changes. While the number of nodes in the subtree increased when the total nodes in the tree increased, the difference between the estimated damage and the actual damage did not change much. This is because even though the graph becomes bigger, the variables that affect how far the damage progresses and the difference between the estimated and actual damage are not changed.

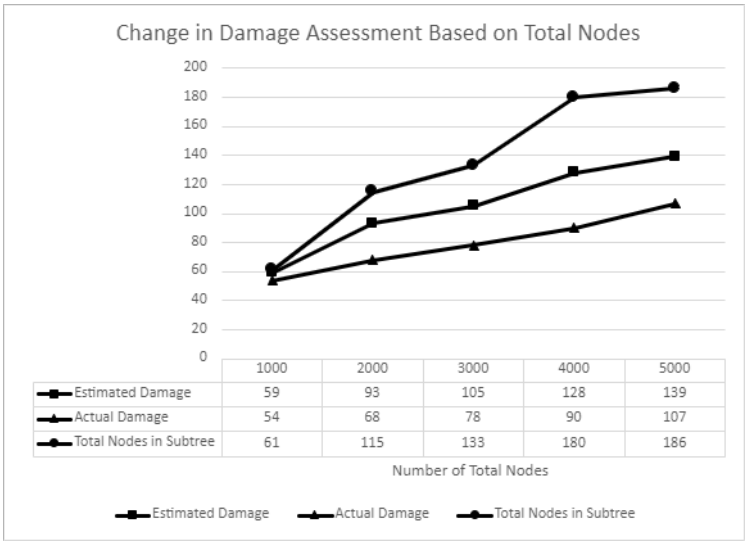

Figure 11a. Change in damage assessment based on total nodes in the system

Experimenting on total time elapsed since the initial damage also had a predictable outcome as indicated by Figure 11b. When there is more time for the damage to spread, both the estimated and actual damage get closer to the total nodes in the subtree. The three lines in the figure eventually converge when enough time elapses for the damage to reach the bottom of the subtree.

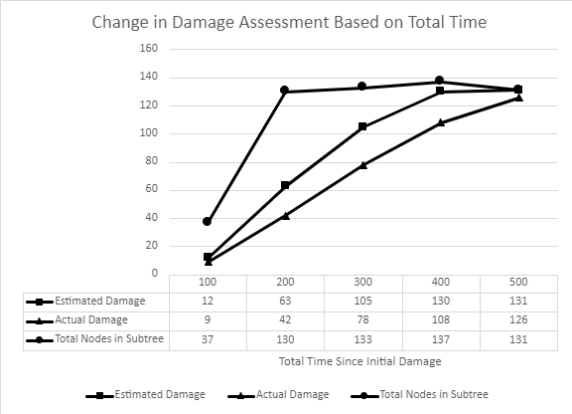

Figure 11b. Change in damage assessment based on the total time since initial damage

Changing the minimum frequency of edges had a more profound effect on the estimated damage specifically as shown in Figure 11c. When the minimum frequency was increased, the estimated damage was more likely to match the actual damage. This can be attributed to a smaller range causing the estimates to match the actual damage.

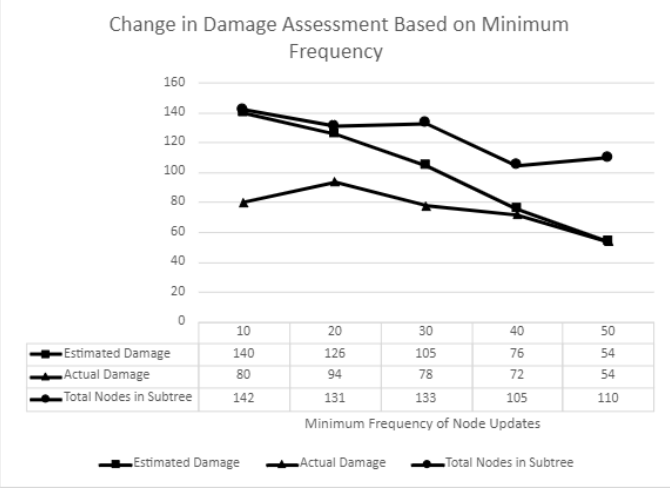

Figure 11c. Change in damage assessment based on the minimum frequency of node updates

In Figure 11d., increasing the maximum frequency of edges resulted in less of an obvious change as increasing the minimum frequency, but it did cause the estimated damage to widen the gap between it and the actual damage. However, when the difference between the

minimum and maximum frequencies becomes high enough, the difference between the estimated and actual damage appears to stabilize even as the maximum frequency increases.

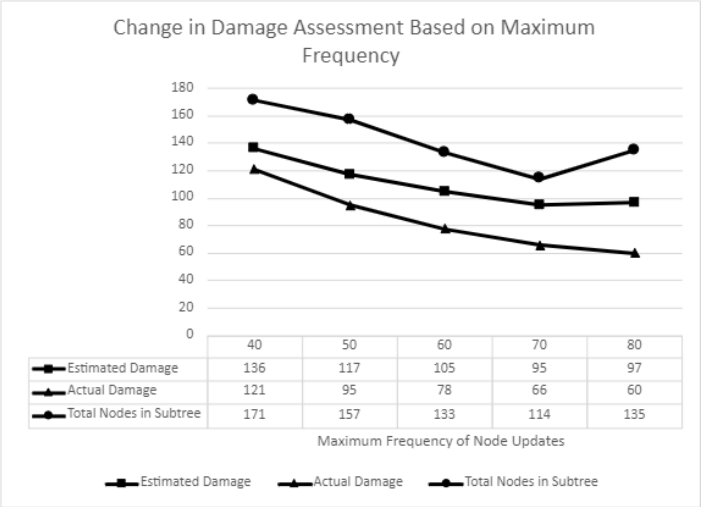

Figure 11d. Change in damage assessment based on the maximum frequency of node updates

However, the most interesting results are in figures 11e. and 11f., as both resulted in changes that seemed to be exponential. This is because changing the maximum number of child and parent nodes changes the overall shape of the graph. When there are more child nodes and less parent nodes, damage can spread to more nodes in less time, because the graph descends in a "horizontal" fashion. Similarly, more parent nodes and less child nodes results in a graph that descends "vertically", and there are less nodes to spread downward to. This causes the two graphs measuring these changes to mirror each other. Furthermore, a low number of either maximum children or parent nodes led to a higher effect on the graph's shape than a higher number of one variable.

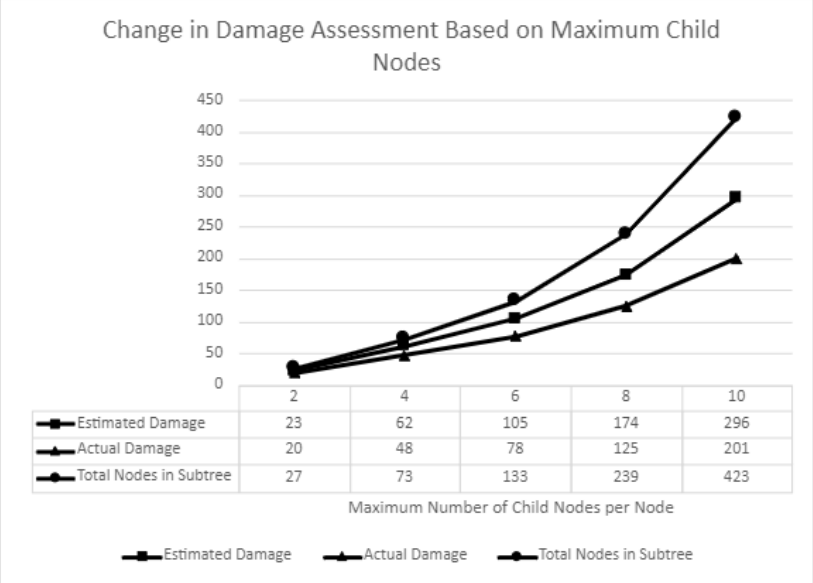

Figure 11e. Change in damage assessment based on the maximum number of child nodes per node

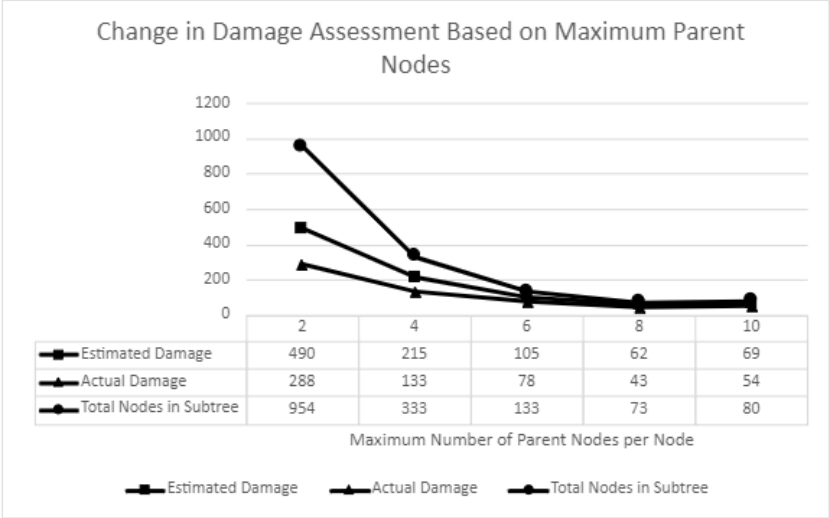

Figure 11f. Change in damage assessment based on the maximum number of parent nodes per node

9 CONCLUSION

This thesis has presented a method to repair data objects that prioritizes quick recovery for the most important components of a system. This allows for the partial restoration of functions during the recovery process with an emphasis on restoring service to the most necessary functions. This was first done by building out three graphs to represent the entire system, what changes the system

made after an attack, and the cascading damage as a result of those changes. Next, an algorithm was developed to optimally schedule repairs by using those graphs to find damage paths that affect the most critical nodes of a system and calculate the fastest repair order to fully restore those nodes. Results from a simulation of this algorithm and the effects on changing the parameters of the damage paths showed how optimally planning repairs can allow for critical nodes to be restored at a fast rate. This work is most applicable to protecting critical infrastructure systems where services need to be restored as quickly as possible to avoid economic or societal disruptions. Additionally, this thesis presented a second model in which damage assessment was done using time-based frequency metrics. This allows for more precise assessments which can result in more services being made available before recovery begins while still ensuring that no damaged systems are used. Experiments using this method showed that the shape of the data system influenced the accuracy of the model more than other factors.

Further work includes integrating these two methods for a more fluid assessment and recovery operation when an attack occurs. Examples include considering the potential for cascading damage from edges with faster update times when determining the criticality of a node, or prioritizing subtrees with the most critical nodes for damage assessment when an attack is detected. Additionally, a method to select the order of repairs for non-critical objects after all critical objects have been repaired is needed.

REFERENCES

[1] I. Narayanan, A. Vasan, V. Sarangan, J. Kadengal and A. Sivasubramaniam, "Little Knowledge Isn't Always Dangerous—Understanding Water Distribution Networks Using Centrality Metrics," in IEEE Transactions on Emerging Topics in Computing, vol. 2, no. 2, pp. 225-238, June 2014, doi: 10.1109/TETC.2014.2304502.

[2] E. Viganò, M. Loi. and E. Yaghmaei, "Cybersecurity of Critical Infrastructure", In Christen M., Gordijn B., Loi M. (eds), The Ethics of Cybersecurity, The International Library of Ethics, Law and Technology, vol 21, Springer.

[3] *Critical Infrastructure Security*: https://www.dhs.gov/topic/critical-infrastructure-security. [retrieved: October 2021]

[4] D. E. Sanger and N. Perlroth, (2021, May 14). "Pipeline Attack Yields Urgent Lessons About U.S. Cybersecurity", https://www.nytimes.com/2021/05/14/us/politics/pipeline-hack.html. [retrieved: October 2021]

[5] A. Anastasios, "Is the Electric Grid Ready to Respond to Increased Cyber Threats?", https://www.tripwire.com/state-of-security/ics-security/electric-grid-ready-increased-cyber-threats/. [retrieved: October 2021]

[6] B. Barrett, "An Unprecedented Cyberattack Hit US Power Utilities", https://www.wired.com/story/power-grid-cyberattack-facebook-phone-numbers-security-news/. [retrieved: October 2021]

[7] K. O'Flaherty, "U.S. Government Issues Powerful Cyberattack Warning As Gas Pipeline Forced Into Two Day Shut Down https://www.forbes.com/sites/kateoflahertyuk/2020/02/19/us-government-issues-powerful-cyberattack-warning-as-gas-pipeline-forced-into-two-day-shut-down/#5f3061645a95. [retrieved: October 2021]

[8] M. Lewis, "Cyberattack Forces Gas Pipeline Shutdown", https://www.jdsupra.com/legalnews/cyberattack-forces-gas-pipeline-shutdown-76217/ [retrieved: October 2021]

[9] B. Panda and J. Giordano, (1999) Reconstructing the Database After Electronic Attacks. In: Jajodia S. (eds) Database Security XII. IFIP — The International Federation for Information Processing, vol 14. Springer, Boston, MA.

[10] S. Patnaik and B. Panda, (2003). Transaction-Relationship Oriented Log Division for Data Recovery from Information Attacks. Journal of Database Management, 14(2), pp. 27-41.

[11] P. Kotzanikolaou, M. Theoharidou, and D. Gritzalis, (2013) Cascading Effects of Common-Cause Failures in Critical Infrastructures. In: J. Butts and S. Shenoi (eds) Critical Infrastructure Protection VII. ICCIP 2013. IFIP Advances in Information and Communication Technology, vol 417. Springer, Berlin, Heidelberg.

[12] J. Ding, Y. Atif, S. Andler, B. Lindström, and M. Jeusfeld, (2017). CPS-based Threat Modeling for Critical Infrastructure Protection. ACM SIGMETRICS Performance Evaluation Review. 45. pp. 129-132. 10.1145/3152042.3152080.

[13] E. Canzani, H. Kaufmann, and U. Lechner, (2016). Characterising Disruptive Events to Model Cascade Failures in Critical Infrastructures. 10.14236/ewic/ICS2016.11.

[14] D. Rehak, J. Markuci, M. Hromada, and K. Barcova, "Quantitative evaluation of the synergistic effects of failures in a critical infrastructure system", International Journal of Critical Infrastructure Protection, Volume 14, 2016, pp. 3-17, ISSN 1874-5482.

[15] N. Bartolini, S. Ciavarella, T. F. La Porta, and S. Silvestri, "Network Recovery After Massive Failures," 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2016, pp. 97-108.

[16] S. Ciavarella, N. Bartolini, H. Khamfroush, and T. Porta, (2017). "Progressive damage assessment and network recovery after massive failures," IEEE INFOCOM 2017 – IEEE Conference on Computer Communications, 2017, pp. 1-9.

[17] J. Štoller and P. Dvořák, "Basic Types of Critical Infrastructure Objects," 2019 International Conference on Military Technologies (ICMT), 2019, pp. 1-7, doi: 10.1109/MILTECHS.2019.8870031.

[18] D. Rehak, P. Senovsky, and M. Hromada, "Analysis of Critical infrastructure Network. Z. Chan, m. Dehmer, F. Emmertsteib, and Y. Shi, *Modern and interdisciplinary problems in network science: a translation research perspective.* 1. Boca Raton, FL, USA: CRC Press, 2018, s. 143-171. ISBN 978-0-8153-7658-3.

[19] Luskova, Maria & Titko, Michal & Leitner, Bohuš. (2016). Multilevel Approach to Measuring Societal Vulnerability due to Failure of Critical Land Transport Infrastructure.

[20] Béla Genge, Piroska Haller, István Kiss, A framework for designing resilient distributed intrusion detection systems for critical infrastructures, International Journal of Critical Infrastructure Protection, Volume 15, 2016, Pages 3-11, ISSN 1874-5482, https://doi.org/10.1016/j.ijcip.2016.06.003.

[21] J. Burns, B. Panda, T. Bui, "Modeling Damage Paths and Repairing Objects in Critical Infrastructure Systems", The Fifteenth International Conference on Emerging Security Informations, Systems and Technologies SECUREWARE 2021, Nov. 14, 2021-Nov. 18, 2021, Athens, Greece, pp. 88-93, ISBN 978-1-61208-919-5