Data Integrity Preservation Schemes in Smart Healthcare Systems That Use Fog Computing Distribution

Abdulwahab Alazeb 1,2,*; Brajendra Panda 2; Sultan Almakdi 1; and Mohammed Alshehri 1,2

1 College of Computer Science and Information Systems, Najran University, Najran 61441, Saudi Arabia; saalmakdi@nu.edu.sa (S.A.); msalsheb@uark.edu (M.A.)
2 Department of Computer Science and Computer Engineering, University of Arkansas, Fayetteville, AR 72701, USA; bpanda@uark.edu
* Correspondence: afalazeb@uark.edu

Abstract: The volume of data generated worldwide is rapidly growing. Cloud computing, fog computing, and the Internet of Things (IoT) technologies have been adapted to compute and process this high data volume. In coming years information technology will enable extensive developments in the field of healthcare and offer health care providers and patients broadened opportunities to enhance their healthcare experiences and services owing to heightened availability and enriched services through real-time data exchange. As promising as these technological innovations are, security issues such as data integrity and data consistency remain widely unaddressed. Therefore, it is important to engineer a solution to these issues. Developing a damage assessment and recovery control model for fog computing is critical. This paper proposes two models for using fog computing in healthcare: one for private fog computing distribution and one for public fog computing distribution. For each model, we propose a unique scheme to assess the damage caused by malicious attack, to accurately identify affected transactions and recover damaged data if needed. A transaction-dependency graph technique is used for both models to observe and monitor all transactions in the whole system. We conducted a simulation study to assess the applicability and efficacy of the proposed models. The evaluation rendered these models practicable and effective.

Keywords: fog computing; IOT medical devices; data integrity; malicious transactions; damage assessment; smart healthcare systems

1. Introduction

Cloud computing has become an attractive computing environment to users and companies since it delivers on-demand services for an affordable cost. Besides, users will be provided massive storage capacity and high computation services with high availability, scalability, and affordability. Despite the cloud advantages, there are different obstacles in the cloud computing environment varying from long response times to security issues. As a solution to cloud computing performance related issues, e.g., high response time and real time data processing [1], CISCO introduced fog computing in 2012, as multiple clouds are deployed close to the ground. This promising technology (i.e., fog computing) enables end users to utilize computing and processing services at the edge of the network [2], which in turn reduces the response time and the computation overhead to the cloud data centers. Additionally, many studies predicted that the number of connected IoT devices would be increased to exceed billions of devices. This has motivated the industry of Internet of Things (IoT) devices to grow and deploying high volume of IoT devices in the market, which results in producing a huge data to be processed. In the fog computing environment, the IoT devices push high volumes of data to nearby fog nodes for analysis and processing then the processed data shall be flushed to the cloud for further processing and storage.

Given that fog computing is considered as an extension of the cloud computing model, it inherits the same security and privacy issues from the cloud. Organizations adapt the
fog computing paradigm by deploying multiple fog nodes to serve an organization’s objectives. However, customers (i.e., individuals and organizations) are concerned about their data security against any malicious actors when they are process in the fog nodes [1]. Although adapting fog computing to intelligent systems would reduce load on the cloud and optimize the processing time and energy consumption as well [3]. Employing security systems that are effective in damage assessment and recovery is required to protect systems from attacks. Therefore, entities can install these security systems by having a trusted fog node management and monitoring transactions in multiple fog nodes in the entire organization when there are public and private fog nodes. However, when the organization has only public fog nodes, it becomes very difficult to have a trusted fog node since public nodes are vulnerable for attacks, and out of organization’s control in this case.

Therefore, in this research, we focus on how sensitive data can be protected from being maliciously modified while being stored or transmitted to fog nodes. We proposed two models to protect against data integrity violation by assessing damage and performing data recovery in case an attack has targeted a fog node(s). We have proposed a system in [4], and can be used in healthcare systems that use fog computing technology to manage and control data across the entire system. The system is based on having a trusted fog node, a private fog node, to manage and monitor all transactions in multiple fog nodes, i.e., private fog nodes, in the entire organization. As an extension for this work, in this paper, we implement, test, and evaluate this model. Then we proposed a new model where we consider the case that the health organization has only public fog nodes which makes it impossible to have a trusted fog node. This model proposes a transaction-dependency graph technique that allows the observation and monitoring of all transactions in the entire system without having a trusted fog node. We proposed different strategies for damage assessments in this architectures. A simulation to analyze the applicability and efficacy of the proposed model is also conducted. Then we compare the performance of the proposed models to measure different metrics like delay and space overhead.

The rest of the paper is organized as follows: Section 2 introduces the literature review, Section 3 explains the proposed scheme, and Section 4 provides the simulation and performance analysis. Finally, the conclusion is in Section 5.

2. Literature Review
2.1. Security and Privacy Issues in Fog Computing

Cloud computing may not help to resolve security and privacy issues, such as data protection, data availability, authentication, and user communication, leaving the role to fog computing [5]. The combination of crucial data enhances privacy and security because most of the data are processed locally at the edge of the network [6]. Security is enhanced as the distance of the data sent is minimized, making fog computing systems advantageous. The local processing and minimization of that distance minimizes the transmission of sensitive data over the network, hence reducing the susceptibility to eavesdropping [7]. Several security and privacy issues can be mitigated by integrating fog computing with IoT infrastructure [8].

However, security and privacy issues have largely contributed to the research and contributions of novel concepts and improvement solutions [5,9]. The utilization of fog computing in smart grids, cities, and intelligent systems such as healthcare and government systems to enhance the quality of services provided, and supply security and privacy to consumers has also attracted the attention of researchers [1,10–14].

A fog computing architecture was mentioned in [15], in which the cloud and IoT have to be provided with end-to-end security. Fog computing architecture relies on fog nodes responsible for managing data and providing communication services in the system. The fog node design should encompass functional security measures to provide reliable security and protection, and achieve a dependable end-to-end computing infrastructure. The establishment of trusted fog nodes means that a safe network can be placed on top of the node infrastructure. This method leads to the formation of a basis for security between
one node and another, a node to a thing, and finally the connection between a node and a cloud.

Zhu et al. proposed a scheme for enhancing privacy by using methods such as blind signature that would ensure anonymity in the authentication processes using the set conditions in the system [10]. Billing problems in smart cities would be resolved by their recommended encryption methods to aggregate smart meter readings in the cloud. This model has its discrepancies that subject customer data to susceptibility to insider and electronic attacks. Silva et al. [16] proposed a general guideline to enhance privacy in IoT environments.

Lyu et al. also addressed the element of smart cities and smart meter readings [11] by suggesting a new framework for aggregating smart meter reading safely through fog nodes to the cloud. The proposed framework involves the addition of statistical noise that is simply irregularities in the data to enhance the data privacy of their clients. The specific technique applied here is the Gaussian noise technique to enable the encryption of data and attain customer privacy.

2.2. Fog Computing in Healthcare

Fog computing has been widely applied in the healthcare industry, which has attracted the attention of numerous researchers [17,18]. Azimi et al. [17] presented a new hierarchical computing architecture of monitoring systems for patients based on IoT to benefit from fog and cloud computing by facilitating the partitioning and executing machine learning data analytics. A gateway, which acts as a bridging point between the sensor infrastructure network and the Internet, is needed for the IoT-based healthcare systems to work effectively according to Amir et al. [19]. To achieve all these, sensor nodes have been utilized to collect data traces on patient movement, utilizing body area networks that are transferred using fog gateways that help provide quick services in medical emergency situations at low latency.

Akrivopoulos et al. [18] designed a smart-phone-based application that would help gather ECG signals from the patient, where the smartphone would act as a fog node. In this case, the patient has maximal control over his health data and can distribute the information to his doctors for health status monitoring purposes. Vora et al. [20] devised a new structure of using fog computing to monitor patients for ambient assisted living.

Vijayakumar et al. [21] described the use of fog computing in the detection and prevention of diseases such as mosquito-borne illnesses. This application was achieved through smart wearable devices or sensors that collect information that are later analyzed and shared through fog computing. Vijayakumar et al. recommended a fog-based health monitoring and risk assessment system that can be applied to differentiate mosquito-borne diseases and create alerts whenever an emergency arises. This system comprises a cyber space, where data processing is undertaken, and a physical space, which contains the user’s information and environmental factors.

Smart cities and grids have been encompassed and have relied on fog computing to be adopted effectively. Naranjo et al. [22] devised a new architecture for the utilization of fog computing in smart cities. The recommended architecture can run the applications on IoT devices jointly for functions such as computing, routing, and communicating with one another through the smart city environment. This architecture decreases latency, and improves the provision of energy and the efficiency of services among things with diverse capabilities.

Tang et al. [23] also made provisions that smart cities would require a new computing paradigm to drive IoT services and applications. They recommended a hierarchical distributed fog computing architecture to allow or support the incorporation of numerous infrastructure constituents and services in forthcoming smart cities.

To advance the smart city concept, Amaxilatis et al. [24] developed an application for smart water metering that would supply data in real time such as consumption on demand as well as bidirectional to end users from metering devices. This application enhances the infrastructure in the concept of smart cities through fog computing.
Aazam et al. [13] discussed the architecture of industrial IoT, which can be described as the use of the IoT in the manufacturing industry for applications such as smart sensors, actuators, and robots. Finally, smart homes have been on the rise according to Froiz et al. [25] owing to technological advancements such as fog computing, which assists in the development of IoT applications. Technologies such as WiFi and ZigBee need to be used for these smart homes to communicate with IoT nodes as well as the cloud. Fog computing must be a solution that provides essential support closer to the end users to ensure local, real-time processing for sensitive, complex tasks.

A distributed fog computing architecture coordinator was proposed in [26] for IoT applications in the smart grid. The key objective of this fog computing coordinator is to occasionally collect information of fog computing nodes, such as information on the remaining resources and tasks. Job management is also achieved through the fog computing coordinator such that all computing nodes can work together on complex tasks. A programming model for fog-based architecture was also proposed. The introduction of fog node coordination is the major difference between their proposed architecture and the traditional one. Fog node coordination aims to enhance the collaboration among fog nodes to meet different requirements in the smart grid.

2.3. Data Integrity in Fog Computing

Ensuring data integrity in fog computing databases is one of the critical topics that still needs more attention from academic researchers; this includes necessary research into the effects of a malicious attack and how to securely and efficiently assess and identify damaged data and recover the data to a state as if no attack happened. Fog computing, as any other modern database infrastructure, still lacks academic contribution to damage assessment and data recovery schemes. Most existing schemes assess damaged data and then recover it only in traditional databases. Damage to data is usually caused by an accident of human error or by malicious acts and attack. Recovery is a rolling back of transactions to revert the database to its previous normal state. This approach should be undertaken immediately after databases are affected to reduce denial of service as well as to ensure the integrity of the data. The accuracy of the algorithms used to assess the damage and recover the databases is in question.

In general, we can divide academic contributions to damage assessment and data recovery schemes in traditional databases into categories based on the technique used. Some used the log files as the only available resource to track dependencies. This technique has two primary pros; it does not incur any additional cost, and it is the most accurate technique, albeit time consuming. Academic research led to mechanisms of damage assessment and data recovery such as follows:

Zuo and Panda [27] introduced two dissimilar methods for the detection of transactions impacted by a malicious attack in a distributed database system. The first method utilized the peer-to-peer model, which is most useful when assessing a single failure point in the system, rather than multiple failure points. The second method is a centralized model whose efficiency is high when used in a large-scale distributed database system; this comes because of the minimization of network communications among the sites.

Liu and Yu [28] intended to advance the efficiency of damage assessment and repair in distributed database systems. First, they identified the challenges and complications faced by those systems. Then, they proposed an algorithm for distributed damage assessment and repair. A local damage assessment and recovery (DAR) was adopted on each site. Later, they adopted an Executor to scan the local log to detect and clean any sub-transaction affected by a malicious transaction. Additionally, a local DAR manager on each site cooperates with the Executor to guarantee global coordination between all sites in the system through the generation of a coordinator for any cleaning transactions.

Other works classify the log operations to multiple groups according to their dependency. So when identification of damaged data items is necessary, then the scheme performs only on the damaged portions of the database. One pro point of this technique is
its ability to determine which items are previously affected. However, this technique can experience of growth in the size of the clusters and dependent transactions may belong to different clusters. Examples of these mechanisms follow: Panda et al. [4] recommended the data dependency method instead of transaction dependency. Every read and write operation of a transaction is classified into one of five different types grounded on the data dependency between each operation. A directed graph, its function characterized by offering up data items that have been affected in the database, is utilized.

Ammann et al. [29] also introduced algorithm sets and recommended a mechanism that would only work on the damaged portions of the database to restore the log files and perform data recovery algorithms immediately after damage assessment. These algorithms only operate while the database is available during repair, but the database must be unavailable during repair especially when the initial algorithm is performing. This approach also offers offline analysis of databases. The process provides data for the repair of damaged transactions.

Some works proposed an extra auxiliary structure for tracking dependencies. The pro of this technique is its ability to repair damaged data without assessing the log file. However, a significant amount of space is required for the preponderance of these mechanisms and they require a substantial amount of unnecessary work.

Mechanisms of this order can be found in the following literature: The column dependency-based approach presented by Chakraborty et al. [30] deduces the relationship between transactions to determine which transactions affected by malicious attacks need to be recovered. In this approach the recovery of data following an attack, which is usually time consuming, requires less time than traditional approaches. Chakraborty et al. suggested a recovery method that would take the affected transactions as input and implement the recovery in two stages: compensation and re-execution. They deduced from their experiments that when malicious transactions increase in the database, the second stage of their recovery scheme also increases.

Rao et al. [31] proposed a scheme for data recovery in a database. The authors created transaction dependencies using application metadata. This scheme only recovers affected transactions. However, it lacks detection of malicious transactions.

Haraty et al. [32] proposed an algorithm that tracks transactions that read from one another and then keep this information in a single matrix. The advantage of this approach is that time is not wasted, and recovery is fast, unlike the traditional methods that would roll back all transactions up to the end. The use of a single 2D matrix helps store dependencies between transactions by identifying the affected segment of the database. Kaddoura et al. [2] recommended also the use of a single matrix for damage assessment and recovery of algorithms.

In [33], approaches were offered by the authors for data recovery that is maliciously attacked through the addition of Before-Image Tables (BI Tables). These BI Tables cannot be modified by any user at any time and have values of all deleted and updated data items. The old value from the BI Tables is rolled back whenever the system detects an update to a data item made by a malicious attacker. They claim that this approach can trace the data as they spread through different machines. The BI Tables are utilized to repair damaged data without assessing the log file.

To the best of our knowledge, no schemes have been developed to address damage assessment and data recovery in the context of fog computing. The first works to address such issues in fog computing were presented in [1,14,34]. Authors of these proposed schemes scanned entire log files beginning with the onset of the attack. This scanning process is followed by communication among fog nodes to detect malicious transactions and attempt recovery. In [1], the authors designed two fog computing architectures in healthcare to deal with damage assessment and data recovery in terms of homogeneous and heterogeneous data. The authors developed a unique technique to deal with damage assessment in each architecture. The proposed algorithm succeeded in damage identification and data recovery. The identified damage is retained for future scans.
In [14], the authors proposed a fog computing–based scheme to deal with utilities management and customers’ data in smart cities. The authors focused on detecting and assessing affected data items. They developed a novel technique to recover the original data and ensure database consistency. A damage audit table was developed to collect the required data for the recovery process.

In [34], the authors proposed a fog computing–based novel approach to secure database integrity in intelligent government systems. In this approach the authors used a fog computing paradigm to secure communications among all a system’s entities. The authors introduced unique algorithms to protect against system integrity violations. In this scheme, the authors created a global graph in the trusted fog node of all transactions in other fog nodes. However, we can still modify the transaction-dependency graph scheme to enable observation and monitoring of all transactions in the entire system in a model that lacks a trusted fog node.

A comprehensive comparison between some proposed schemes on the literature reviews is shown in Table 1.

| Type of Dependency | Space Requirement | Auxiliary Structure | Methodology | Drawback |
|--------------------|--------------------|---------------------|-------------|----------|
| [27] Transaction   | NA                 | NA                  | Scan the entire log files. | Time consuming. |
| [2,32] Data        | Large Space        | Matrix              | Scan only the matrix.     | Require a lot of space. |
| [33] Data          | Large Space        | Tables              | Refer to tables           | Require space and Do unnecessary work. |
| [1] Data           | NA                 | NA                  | Scan the affected log files. | Continuous communication between all affected fog nodes. |
| [14] Data          | NA                 | NA                  | Scan the affected log files. | Continuous communication between all affected fog nodes. |
| The Proposed Scheme| Transaction        | Small Space         | Graphs                  | Scan only the affected graphs. |

3. Proposed Scheme

Generally, there are three types of fog computing technology: public fog computing, private fog computing, and mixed fog computing combining public and private. Any distributed architecture that offers general services for public users and consumers is considered a public fog computing network and any distributed fog computing network to which access is limited and restricted for use only by specific parties is a private fog computing network. Great examples of private fog computing networks are the fog computing distributions for school, college, and government campuses [35].

In previous work [34], we proposed a scheme using a unique private distributed fog computing architecture to manage important data for an intelligent county government system. We also introduced a cooperative method for identifying damaged and affected transactions following a malicious attack. In this paper we propose two unique models to manage the data in a healthcare environment system that uses fog computing: one model for private fog computing distribution which uses the same technique as [34] and the other model for public and mixed fog computing distribution.

For each one we present an effective technique that employs a transaction dependency graph to detect and assess damage sustained by data and transactions affected by an attack. In the proposed models we assume that the Intrusion Detection System (IDS) is responsible for detecting malicious transactions in the system and providing a copy of those transactions to the proposed damage-assessment algorithms. Each fog node in the proposed system must have its own log file and is required to use a strict serializable history. The log files must be secured such that they cannot be modified by unauthorized individuals.
3.1. Scheme Abbreviation Part

A description of the abbreviation to be used in our scheme can be found in Table 2.

Table 2. Abbreviation used in our scheme.

| Notation       | Definition                                                                 |
|----------------|---------------------------------------------------------------------------|
| FN\textsubscript{pub} | Public fog data service node on the system.                              |
| MSFN           | The main medical service fog node.                                        |
| S[log\textsubscript{ID}.T\textsubscript{M}] | The detected malicious transactions set done by IDS.                     |
| G(T\textsubscript{n}, E)  | Graph representation where T\textsubscript{n} indicates the number of transactions in the node and E indicates the number of edges. |
| Aff-L\textsubscript{fog}\textsubscript{n} | List of all affected transactions that have been identified by the proposed mechanism. |
| T\textsubscript{No.}   | Transaction which is a single unit of logic in database and contains of multiple operations |
| A, X, or W      | Data item which is the smallest element in the transaction.               |
| w\textsubscript{i}(A, v\textsubscript{1}, v\textsubscript{2}) | The write operation of the transaction T\textsubscript{i}; v\textsubscript{1} is the old value of the data item A, and v\textsubscript{2} is the new value of data item A after it is updated. |
| r\textsubscript{i}(A, v) | The read operation of transaction T\textsubscript{i} where A is the data item and v is the current value of A. |
| c\textsubscript{i}     | The transaction T\textsubscript{i} has been successfully committed to the database. |
| O               | Operation \in T\textsubscript{i} (write, read, or committed).            |
| \Theta(V + E)   | \Theta is Big-O notation, which symbolizes upper bound of the space or running time. V is the set of vertices which in our models is the number of transactions in the log file T, and E is the set of edges which are the dependencies between two transactions. |

3.2. First Model

In the proposed private fog computing distribution model, we assume that the whole distribution is trusted since it is private distribution and owned by the same healthcare organization. The organization has several medical clinics within a certain geographic range which could be city or county. Therefore, each medical clinic in this organization operates its own fog nodes distribution system. Each system has at least one main medical service fog node (MSFN) and many edge nodes.

The edge nodes are responsible for collecting data using the IoT and smart devices such as medical wearable, smart thermometers, and patient monitors. Then the data is transferred to the MSFN. Each MSFN in the system can perform all essential operations on collected data. These necessary operations optimize the network’s bandwidth since the aggregation of data sent through it is accomplished by force.

Essential data that is aggregated increases privacy and security because the preponderance of the data is processed locally, at the edge of the network; local processing reduces and minimizes sensitive data transmission over the network.

For instance, the pediatric clinic has one MSFN and several edge nodes. The MSFN is responsible for controlling and managing patient data. Each edge node is responsible for collecting patient data such as heart-rate, body temperature, etc. All data collected by smart thermometers or wearable sensors are transferred through edge nodes to the main MSFN in the system. Likewise, certain other system MFSNs, such as the urgent care clinic’s MSFN, have restricted access to the data collected by other edge nodes or system MFSNs.

As stated earlier this is a private distributed fog computing network model so the whole system is assumed to be trustworthy as the same organization operates and surveils it. However, one trusted system fog node manages security matters including key management and distribution. This trusted fog node is a leader in damage assessment and data recovery processes (Figure 1).
The Proposed Damage Assessment Algorithms

The entire system’s transactions in this model are observed and monitored by a transaction-dependency graph. All transactions affected by a malicious transaction can be quickly detected by a transaction-dependency graph. Transaction dependency is used instead of a more complicated data dependency. Data dependency poses difficulty in tracking transaction updates and graphs are sparser and less visible. A transaction-dependency graph is easier to build and can detect and block malicious transactions considerably faster, consequently preventing further damage to system data. Data items unaffected by malicious transactions are available for immediate use.

The subsections that follow detail each step of this model.

- **Algorithm 1**: building local graphs for private distributed fog computing:

Each primary MSFN constructs a directed graph based on transaction-dependency relationships between transactions that have been successfully committed to its database. The graph is produced and stored in a separate, unmodifiable file with the original transactions in the same directory of the log file.

Each MSFN generates a graph built in a two-row matrix. Associated data items are contained in the first row and the last transaction that updated each of those data items is contained within the second row (Tables 3 and 4). When \( T_i \), a new transaction, is committed to the database, the matrix is scanned to determine \( T_i \)’s precursors (parents). \( T_i \) is added to the graph as a new vertex and, based on those dependencies, the edges between \( T_i \) and its parents are inserted. Concurrently, the newly committed transaction is updated in the matrix. So, for each write operation for data item \( X \) in \( T_i \), the algorithm scans the matrix and updates the last transaction for that data item. If, however, the matrix does
not include data item $X$, it will insert, as a new record, that data item and the transaction identification $T_i$.

**Algorithm 1** Building Local Graphs.

1. Create a new Hash-table $H$ and Initialize to null
2. Create a new Graph Representation $G = (T_i, E)$ and Initialize to null
3. When $T_i$ is committed in the local DB then
4. for each operation $O \in T_i$ do
5. if $O$ is $w_i (A, v1, v2)$ then
6. add a pair $(A, T_i ID)$ to $H$
7. add $T_i$ as a new vertex to $G$ if it does not exist
8. else if $O$ is $r_i (A, v)$ then
9. if $A \in H$ then
10. add $T_i$ as a new vertex to $G$ if it does not exist
11. acquire the parents $T_i$ ID from $H$
12. add edge from $T_i$ to $T_i$
13. else if $A \in$ foreign fog node then
14. call global graph algorithm (Algorithm 2)
15. end if
16. else if $O$ is $c_i$ then
17. end if
18. end for

Table 3. Matrix tracking the last updated transactions after $T_5$ is committed.

| Data Items | C     | G     | A     | D     |
|------------|-------|-------|-------|-------|
| Last updated $T$ | $T_1$ | $T_4$ | $T_4$ | $T_5$ |

Table 4. Matrix tracking the last updated transactions after $T_{10}$ is committed.

| Data Items | C     | G     | A     | D     | B     | E     |
|------------|-------|-------|-------|-------|-------|-------|
| Last updated $T$ | $T_9$ | $T_4$ | $T_6$ | $T_6$ | $T_6$ | $T_{10}$ |

For example, consider the following log records for MSFN$_x$ and MSFN$_y$:

MSFN$_x$: $r_1(A, 45) r_1(B, 34) w_1(C, 29, 80) w_1(G, 23, 18) r_2(B, 34) c_1 w_2(A, 45, 48) w_2(D, 12, 35) c_2 r_3(G, 18) c_3 w_3(A, 48, 69) w_2(G, 18, 24) c_4 r_3(D, 35) r_5(A, 69) w_3(D, 35, 98) c_5 r_6(B, 34) w_4(B, 34, 63) r_6(D, 98) w_3(D, 98, 24) r_6(A, 69) w_4(A, 69, 78) c_6 r_7(D, 24) c_7 r_8(C, 80) c_8 r_9(fog$_x$-T$_6$ TV, 56) w_9(C, 80, 93) c_9 r_{10}(A, 78) r_{10}(C, 93) w_{10}(E, 39, 66) c_{10} r_{11}(E, 66) c_{11}.

MSFN$_y$: $r_1(K, 58) r_1(fog$_y$-T$_4$-G, 24) w_{11}(K, 58, 98)c_1 r_3(M, 39) r_3(K, 98) w_{12}(M, 39, 59)c_2 r_3 (R, 43)c_3 r_4(fog$_y$-T$_5$, D, 35) r_4(M, 59) w_{13}(N, 36, 84) c_4 r_5(K, 98) w_{14}(P, 43, 78)c_5 w_{15}(V, 56) c_6$

The log records for MSFN$_y$ indicate that $T_9$ read data items $B$, $D$, and $A$. After $T_9$ is committed, it is added as a new vertex to the local graph of MSFN$_x$, and the directed-dependency edge (the link) between $T_9$ and other transactions on the graph is drawn based on $T_9$'s dependency relationships.

The algorithm locates the last updated transactions in the matrix for data items $B$, $D$, and $A$ (Table 3). The algorithm finds that $T_4$ is the last transaction that updated data item $A$ and $T_5$ last updated data item $D$. $B$ does not exist in the matrix and so it has not been modified. A directed edge is now drawn from $T_4$ to $T_6$ and from $T_5$ to $T_6$ (Figure 2). The matrix is simultaneously updated. A new record for data item $B$ is inserted and associated with $T_6$ as the last transaction to update $B$. In addition, the last updating transactions associated with data items $A$ and $D$ are updated to $T_6$ as shown on Table 4. Figure A1 in the Appendix A represents the graph of MSFN$_y$. 
Algorithm 2: building global graphs on the trusted fog node:

As the trusted fog node plays an essential role in the management of security and protection issues, including damage assessment and data recovery, the trusted fog node secures global graphs for any transaction that accesses a data item in a system MSFN node.

Here is an example:

- \( T_i \), committed to the MSFN\(_y\) accesses data item \( A \). The last transaction update for data item \( A \) was \( T_j \) on MSFN\(_x\).
- A copy of the graph updating that last transaction’s data item \( A \) is sent to the global graph on the trusted fog node. The global graph includes all preceding vertices that could directly or indirectly affect \( T_j \).
- Transaction \( T_i \) is added as a new child of \( T_j \) to the global graph and an edge is drawn from transaction \( T_j \) to \( T_i \).
- Transaction \( T_i \) from MSFN\(_y\) continues updates to the global graph on the trusted fog node by copying and sending a copy of each subsequent transaction consigned locally at the MSFN\(_y\) (successor of \( T_i \)). This copy updates and is sent frequently.

Algorithm 2 Building global graphs on the trusted fog node.

1. Create a new Global Graph Representation \( G_g = (V, E) \) on trusted fog node and initialize to null
2. (When \( T_i \) ∈ fog\(_x\) reads data item \( A \) ∈ fog\(_y\)) then
3. for each \( r_i (A, v) \in T_j \) do
4. acquire all predecessors of \( T_j \) from \( G_y \)
5. add \( T_i \) as a new vertex to \( G_g \) if it does not exist
6. add edge from MSFN\(_y\).\( T_j \) to MSFN\(_x\).\( T_i \)
7. end for
8. (When a new transaction \( T_z \) ∈ MSFN\(_x\) is committed where \( T_z \) is successor of \( T_i \)) then
9. add \( T_z \) as a new vertex to \( G_g \) if it does not exist
10. add edge from \( T_i \) to \( T_z \)
Figure 3, shows an example of the global graph generated using transaction activities at the trusted fog node from MSFN\_x and MSFN\_y. As seen in the log records of MSFN\_y, T\_1 read data item G, which was updated by T\_4 in the log records of MSFN\_x. Then T\_4, MSFN\_x graph and all its preceding elements are sent to the trusted fog node. This means any vertex that is a precursor of T\_4 are now stored in the trusted fog node. In this case, T\_4 is the only vertex since it does not have parents. In addition, once T\_1 from MSFN\_y is committed, it is added to both the global graph on the trusted fog node and the local MSFN\_y graph.

All new transactions added to the local graph of MSFN\_y and the successor of T\_1 are added to the global graph. Thus:

- In the log records of MSFN\_y, T\_2 reads data item K from T\_1. T\_2 is added as a new vertex in the local MSFN\_y graph.
- An edge is drawn from T\_1 to T\_2. The global graph is subsequently updated by adding T\_2.
- And so, it continues: T\_4 reads item M from T\_2 and T\_5 reads item K from T\_1. Hence, T\_4 and T\_5 are added as new vertices to the local MSFN\_y graph. Edges are drawn from T\_2 to T\_4 and from T\_1 to T\_5. Then T\_4 and T\_5 are added to the global graph. The MSFN\_y will continue sending updated copies of T\_1’s successor to the global graph upon commit operating at its local database. This copy, as indicated, is updated and sent frequently.

To prevent the global graph from becoming excessively large, a threshold is set based on the system’s hardware capabilities. When the number of transactions in each fog node reaches that threshold, the global graph is removed to the cloud for permanent storage.

- Algorithm 3: damage assessment algorithms for private distributed fog computing:

Once the IDS identifies a malicious transaction in the system, it transmits it to the trusted fog node for identification of all transactions dependent on, and consequently affected by, the malicious transaction. Simultaneously, each primary fog node in the system’s MSFN data service node victimized by the malicious transaction receives notice of its detection on the database.
The MSFN scans the graphs it possesses and detects affected transactions. This is a necessary feature as some local graphs contain transactions, malicious or not, never accessed or read by another fog node. Those transactions will not be forwarded to the trusted fog node. The MSFN, using a modified Depth First Search (MDFS) algorithm, scans only the graphs in its possession and begins with the malicious transaction sent and received from the IDS. The MDFS algorithm can efficiently identify all successor transactions of the malicious transaction in the graphs which consider affected transactions.

Succeeding immediately after this portion of the damage-assessment process, each subsystem will receive a list of all transactions affected by the malicious transaction and detected on its local database. All data items updated by those transactions are now considered damaged and are blocked from access until recovered. The remaining data items are made available for users.

**Algorithm 3** Damage assessment algorithm for private distributed fog computing.

```plaintext
1: Once S{MSFN|D,T_M} is received from IDS
2: Classify S{MSFN|D,T_M} to multi-set {S_1[MSFN_1,T_M]...S_n[MSFN_n,T_M]}
3: Send each subset S_n[MSFN_n,T_M] to the (MSFN_n)
4: Send S{MSFN|D,T_M} to the trust fog node.
5: For the trust fog node and each other MSFN_n that receive S or S_n.
6: MDFS (G_n, T_M)
7: Create a new affected-transactions list Aff-Lfog_n and Initialize to null
8: Create a new Stack tranS and Initialize to null
9: add T_M to tranS
10: mark T_M as visited
11: while tranS is not empty do
12: T_M = tranS.pop
13: add T_M to Aff-Lfog_n
14: for each child T_i of T_M in G_n do
15: if T_i is not in then Aff-Lfog_n
16: add T_i to tranS
17: mark T_i as visited and add it to Aff-Lfog_n
18: end if
19: end for
20: end while
```

**3.3. Second Model**

In the second model, we implement a transaction dependency graph technique in a scheme architecture that uses a public or mixed fog computing distribution. The latter combining public and private fog nodes. We observed that this scheme lacks the availability of a trusted fog node to build global graphs and handle security matters such as damage assessment and data recovery. This scheme is a mixture of public fog nodes, FNpub, which belong to more than one owner and private fog nodes, MSFN, which belong to different healthcare providers. No trusted fog node can exist in this distribution type.

This model presents a unique mechanism to assess and detect damaged and affected transactions following a malicious attack. In the proposed scheme, as in the first model, an intrusion detection system is responsible for identifying malicious transactions in the system and providing a copy of those transactions to the proposed damage assessment scheme. Each fog node in the system is required to use a strict serializable history on its own log file.

As seen in Figure 4, this model can be used in a geographic area, such as a city or county, larger than the first model can maintain, since the public fog node can help increase communication between the fog nodes of the healthcare providers utilizing the system.
Each healthcare provider in the system has their own private fog node, MSFN. They have restricted access between fog nodes. Factors determining access include whether or not providers belong to the same organization and whether or not access is granted a provider by a patient’s release of records from one healthcare provider to another. On the other hand, the public fog node can use the IoT medical devices to collect data from patients and send it to the proper MSFN. This point is important in ensuring reliable connections between patients and their healthcare providers. Further, doctors can remotely diagnose and follow up with their patients.

This model is needed, just as well as the first model, as it is more applicable to situations where consumers prefer to use public services to maximize their technology experience. Additionally, there are patients requiring visits to doctors from different healthcare providers.

- **Algorithm 4**: building local graphs for public distributed fog nodes:

  Each public FN\text{pub}\text{, and private MSFN} fog node in the system will create a directed graph based on the transaction dependency relationships among transactions that have been successfully committed to the database. The graph will be kept in a separate, unmodifiable file in the same directory of the log file. To create the graph, each fog node will continue tracking each data item and the transaction that last updated it. This algorithm works similarly to the algorithm for building local graphs in the first model.

  When a new transaction (T_x) is committed to the database, the algorithm will determine T_x’s dependency (parents). T_x will then be inserted as a new vertex to the local graph and the links between T_x and its parents will be added based on the dependen-
cies. The fog node will update the tracking mechanism in accordance with the recently committed transaction.

Algorithm 4 Building local graphs for public distributed fog computing.

```plaintext
1: Create a new Hash-table H and Initialize to null
2: Create a new Graph Representation G = (T, E) and Initialize to null
3: When T is committed in the local DB then
4: for each operation O ∈ T do
5:   if O is w_i (A, v1, v2) then
6:     add a pair (A, T, ID) to H
7:   else if O is r_i (A, v) & & A ∈ H then
8:     add T_i as a new vertex to G if it does not exist
9:     add edge from T_i to T
10: else if T_k ∈ foreign fog node fog, reads data item A that updated by T_j then
11:     add fog, T_k as a new vertex to G if it does not exist
12:     add edge from T_j to fog, T_k
13: end if
14: end for
```

The main difference between this algorithm and Algorithm 1 is apparent when any foreign transaction has accessed a data item from another fog node. However, when any transaction T_i reads a data item in one FNPub/MSFN node that has been updated by T_j in another FNPub/MSFN node, then the transaction identification (ID) of T_j will be added as a child of T_i on the local graph of the FNPub/MSFN node, in which the item is initially updated.

To illustrate the algorithm, we use the following example for the records of log for MSFN1 and FNPub2 fog nodes:

**MSFN1:** r1(X, 32) w1(Z, 29, 80) w1(W, 89, 81) c1 r2(Z, 80) w2(X, 32, 22) w2(D, 12, 35) c2 r3(W, 81) r3(FNPub2→T2→F, 66) w3(W, 81, 91) c3 w4(X, 22, 77) c4 r5(D, 35) r5(X, 77) w5(D, 35, 98) c5 r6(D, 98) w6(D, 98, 24) r6(X, 77) w6(X, 77, 78) c6

**FNPub2:** r1(E, 13) r1(MSFN1→T1→W, 81) w1(E, 13, 39) c1 r2(F, 53) r2(E, 39) w2(F, 53, 66) c2 r3(R, 43) c3 r4(MSFN1→T5→D, 98) r4(F, 66) w4(I, 79, 30) c4 r5(E, 39) w5(I, 43, 78) c5

The log record for MSFN1 indicates that T_j reads data items (D) and (X). Consequently, after T_5 is committed, it will be added as a new vertex to the local graph of MSFN1, and the directed dependency edge between T_5 and other transactions on the graph will be added on the basis of T_5 dependency relationships. The algorithm will determine that the last updated transaction for data item (X) is T_4 and the last transaction that updates data item (D) is T_2. Hence, a directed edge will be drawn from T_4 to T_5 and from T_2 to T_5 (Figure 5).

In the log records of FNPub2, T_1 reads data item W that is updated by T_1 in the log records of MSFN1. The transaction ID of T_1 from FNPub2 will then be added as a foreign child of T_1 on the local graph of MSFN1. T_4 from the log of FNPub2 reads item D that is written by T_5 on MSFN1, and the transaction ID of T_4 from FNPub2 will be added as a foreign child of T_5 on the local graph of MSFN1.

Data item F is updated by T_2 on FNPub2 and accessed by T_3 from MSFN1. Accordingly, T_3 will be added as a foreign child of T_2 on the local graph of FNPub2. The process continues updating local graphs, given that transactions are successfully committed to the local database. Figure 5 shows examples of the local graphs generated using transaction activities at MSFN1 and FNPub2.
• Algorithm 5: damage assessment algorithms for public distributed fog nodes:

**Algorithm 5** Damage assessment algorithm for public distributed fog nodes.

1. Once S{fogID:T_M} is received from IDS
2. Assort S{fogID:T_M} to multi-set S_1 ... S_n
3. Send S_x{fog_x:T_M} to the (fog_x)
4. For each fog_x that receive S_x
5. MDFS (G_n , T_M)
6. Establish a new list for affected transactions (Aff-Lfog_x)
7. Establish a new Stack S_T
8. add T_M to S_T
9. mark T_M as visited
10. while S_T is not empty do
11. T_M = S_T.pop
12. add T_M to Aff-Lfog_x
13. for each child T_i of T_M in G_n do
14. if T_i /∈ Aff-Lfog_x then
15. add T_i to S_T
16. mark T_i as visited
17. add T_i to Aff-Lfog_x
18. if T_i is foreign transaction ∈ fog_y then
19. add T_i to sub-list Aff-Lfog_{x,y}
20. end if
21. end if
22. end for
23. end while
24. Send Aff-Lfog_{x,y} to fog_y to do further detection

Once the IDS detects a malicious transaction in the system, each fog node, FN_{pub}/MSFN, is informed of the malicious transaction(s) detected on its database. The FN_{pub}/MSFN scrutinizes its own graphs and identifies affected transactions. The compromised fog nodes examine only the graphs they possess using a modified Depth First Search algorithm (MDFS) and beginning with the first malicious transaction received from the IDS. All successors of the malicious transaction will be considered affected transactions and added to the affected list. However, if the transaction T_j from FN_{pub2} is found to be a successor of the initial malicious transaction T_j on the MSFN_1, then a sub-list of affected transactions

![Figure 5. Described local dependency graphs for MSFN_1 and FN_{pub2}.](image-url)
will be sent to FN_{pub2} and used as input to the damage assessment algorithm. The process will continue until all affected transactions in the system have been identified.

Soon after this phase of the damage assessment mechanism is accomplished each fog node will have the final list of all affected transactions detected on its local database. Any data items updated by any of those affected transactions will be blocked from access until recovered as they are considered damaged data items. Unaffected data items will remain available to users.

4. Implementation and Evaluation

4.1. Setup

In the experiments, a personal computer with 16 gigabytes of RAM and a Dual-Core Intel Core i7 processor with a speed of 3.1 GHz was used. The whole system environment was simulated using Java to prove the model and algorithms’ applicability and efficiency. The efficiency of the proposed algorithms was tested and evaluated by conducting different experiments considering different factors, e.g., number of transactions in log files, the number of attacking transactions, and fog nodes in the system, to see how they affect the delay in the whole system. Note that, the communication delay between the fog nodes in the experiments has not been measured since we used a local personal computer to perform. Since our work is novel in the context of damage assessment in fog computing, there is no existing work that we can compare the performance of the proposed systems against. For this reason, the aim is to implement, test, and evaluate the proposed models and prove that they are not only applicable but also accurate, scalable, and reasonable in the reported delays. However, we compare the performance of the proposed models against each other to see which model is better for different circumstances.

4.2. Data-Set

4.2.1. Log Files

Since there is no available dataset for damaged transactions’ log files, and this work is novel, We generated random log files, vary in length, for every fog node in the system. The contents of the log files may be different for each model based on given assumptions (i.e., we assumed some of transactions have been modified or tampered with). We performed our experiments based on the three log file sizes beginning with 100 transactions in each log file and following through with 500, and finally 1000. The data dependency between the fog nodes was inserted arbitrarily into the log files. Here is an example of the generated log files: [r_{1}(361,688) w_{2}(345,669) r_{4}(372,607,689) r_{5}(207,651) w_{1}(227,688,688) w_{2}(345,669,621) r_{3}(399,653) r_{4}(282,634) w_{5}(207,651,612) c_{1} c_{2} w_{3}(399,653,635) r_{4}(361,688) w_{5}(374,678,643) w_{3}(279,657,640) c_{4} c_{5} c_{3} r_{6}(238,689) w_{7}(207,612,636) r_{6}(356,641) r_{5}(323,683) w_{6}(238,689,604) w_{7}(262,679,633) w_{6}(356,641,654) w_{9}(323,683,635) w_{6}(345,621,636) c_{7} c_{8} c_{9} w_{6}(286,646,677) c_{6} w_{10}(313,663,685) r_{10}(351,604) c_{10}] Below we explain the notations used for read, write, and commit operations respectively.

- r_{2}(345,669): is reading operation where r_{transaction_number} (data item, value)
- w_{2}(345,669,621): is writing operation where w_{transaction_number} (data item, old value, new value)
- c_{2}: The transaction 2 has been committed which means it is successfully completed.

4.2.2. Graphs

We used the adjacency list for directed graph representation due to its speed and space efficiency technique [36]. In this situation, we represent the transaction T as vertex V in the directed graphs. If there are |T| in the log files, then each list can have up to |T| – 1 transactions depend on the transaction dependency that we explained previously. Each vertex in the adjacency list can be reached in constant time since we need only to refer to an array.

Regarding the space complexity of the adjacency list, it is the best case scenario of graph representation techniques for storing the directed graph in the computer. This
technique will save substantial space. Thus, the adjacency list will only take up to $\Theta(V + E)$ space, where $V$ is the set of vertices which in our models is the number of transactions in the log file $T$, and $E$ is the set of edges which are the dependencies between two transactions [37]. Additionally, the adjacency list allows us to easily insert a new edge or vertex without extra cost as we are using a linked list structure and this representation is more informative and provides easier tracking of any adjacent nodes of node.

All local and global directed graphs were built simultaneously when the log files were generated based on transaction-dependency relationships between the transactions, as previously explained in Section 3. With each log file, two graphs were generated, one for the model where there is no trusted fog node (no global graph) in the system and the other one for the model where there is a trusted fog node in the system. The global graphs for the trusted fog node were also generated for any transaction that accessed a data item from another fog data service node in the system. The log files and graphs were manually rechecked and examined as ground truth to ensure the correctness and accuracy of the algorithms.

4.3. Experiments and Evaluation

We started with a fixed number of fog nodes each time and a different number of transactions in each log file. Each time we randomly inserted a malicious transaction and then classified the results into three different clusters, or sets, of affected transactions. The first set comprised less than five identified, affected transactions. The second set contained 10 to 15 identified, affected transactions, and the third set included 30 to 35 identified, affected transactions. This clustering was important for making a fair and reasonable comparison of the proposed algorithms as they were impacted by other factors such as the number of fog nodes and the number of transactions in each log file. Each transaction in each set was repeated approximately 20 times, and the total time from inserting the malicious transaction until all affected transactions were identified in the system was computed. Then the average for each set was calculated for investigation and evaluation of our approach. Then we compared the results to determine which factor(s) created a greater or lesser impact to the algorithms.

4.3.1. The Impact of the Different Sets of Affected Transactions on Various Fog Nodes Number on the First Model

Figures 6 and A2–A4 show the relationship between the number of affected transactions, the number of transactions on each log file, and the average time required by our system to detect all affected transactions in the whole system. The average time was calculated by averaging the detection time each set of malicious transactions required. The result shows that the number of affected transactions that are detected has great effect on the total time the system remains unavailable. This applies in all cases, regardless of the number of fog nodes or the number of transactions on each log file.

![Figure 6. The impact of the different sets of affected transactions on five fog nodes.](image-url)
4.3.2. The Impact of Different Number of Transactions on Various Number of Fog Nodes on the First Model

Figures 7 and A5–A7 show the relationship between the number of transactions on each log file, and the average time required for our system to detect all affected transactions in the whole system. The result shows that the number of transactions does not have a big impact on the total time the system will be unavailable. This is the case with all other scenarios, either with a different number of fog nodes in the system or a different set of affected transactions.

![Figure 7](image_url)

**Figure 7.** The impact of different number of transactions on five fog nodes.

4.3.3. The Impact of the Different Sets of Affected Transactions on Various Fog Nodes Number on the Second Model

The experiment was undertaken on the second model whereby the system has no trusted fog node. This was attained through the employment of the transaction-dependency graph system on both prototypes. The experiment’s main purpose was to assess the performance of the damage assessment algorithm during the process of detecting the affected transactions in the absence of trusted fog node. As earlier described in Section 4.2.1, the log files created were based on transaction-dependency relationships that existed within transactions and were additionally built at the same time as the local and global directed graphs. For every existing log file, two variable graphs would be produced. One represents the second model that does not have a trusted fog node (and absence of a global graph) in the system utilized in the investigations present in Section 3.3.

In order to enhance validity of the investigation, repetitive tests were conducted to ensure fair comparisons of the outcomes from the two models, as depicted in investigations in Section 4.3 concerning trusted fog nodes. This was achieved by using a static number of fog nodes accompanied by a dissimilar number of transactions in every log file. Then we insert the same malicious transaction that was randomly entered for the first model, whereby the affected transactions also were grouped in three different sets or groups. In the first group, fewer than five affected transactions were contained. The second group comprised about 10 to 15 transactions that were affected, while the third grouping contained about 30 to 35 transactions that were affected and had been recognized. The average for each group was determined in order to examine and appraise our approach. In order to define the factor(s) impact (great or less) on the algorithms, a comparison of the results was conducted.

Figures 8 and A8–A10 illustrate the correlation between the number of affected transactions, the number of transactions in each log file, and the average time required by our system to detect all maliciously compromised transactions in the whole system. The average time was calculated by averaging the detection time each fixed set of undamaged transactions required.
The result shows that the smaller set of less than five affected transactions took an average of 0.076 ms to 0.079 ms for discovery between the variables in log files and fog nodes. That time almost doubled for the set containing 10 to 15 affected transactions: averaging 0.14 ms to 0.15 ms. The final set of 30 to 35 affected transactions, claimed an average of 0.31 ms to 0.39 ms. Therefore, the number of detected, compromised transactions had great effect on the total time the system remained unavailable. This applied in all cases, regardless of the number of fog nodes or the number of transactions held by the log file.

4.3.4. The Impact of Different Number of Transactions on Various Number of Fog Nodes on the Second Model

Figures 9 and A11–A13 demonstrate the relationship between the number of transactions on each log file and the time requirement, averaged, for discovery of all affected transactions in the whole system. Overall, the results indicate that the number of transactions did not have a significant impact on the total delay time which affected system availability. Still, there was some small impact.

For example, in the experiment utilizing the set of less than five affected transactions, the log file of 100 transactions claimed less delay time than the larger log files by almost 0.001 ms to 0.002 ms.

On the other hand, the set of 10 to 15 affected transactions in the log file of 100 transactions demanded more time than the larger log files by 0.01 ms. All other scenarios, either with a different number of fog nodes in the system or a different set of affected transactions, offered similar results.
4.3.5. Overall Comparison between the Two Models

We performed this experiment to show the difference between two models: the model with a trusted fog node (global graph) present in the system and another where there was no trusted fog node in the system (no global graph). Which was the better performing of the two, and under which factors? Figures 10–12 show the overall comparison between these models on the average runtime. The result, illustrated by Figure 10, indicates that the second model, where there was no trusted fog node, performed better, in terms of execution time, with the set of less than five affected transactions in almost all cases by 0.001–0.002 ms. The explanation for that lies in the use of the global graphs. The first model uses the global graphs as input for the algorithm and those graphs are larger than the local graphs that are used as input for the second model. Additionally, the damage caused by the small set of less than five affected transactions usually did not affect many fog nodes. On average only one to two fog nodes were affected. The second model stood out a bit from the first model in this case.

Figure 10. Comparison between having global graph on trusted fog node and not on a set of less than five affected transactions.

Figure 11. Comparison between having global graph on trusted fog node and not on set of 10 to 15 affected transactions.
However, as seen in Figures 11 and 12, the results differed for the two larger sets of affected transactions. We observed that the model with a trusted fog node required less time than the model without a trusted fog node by an average of 0.018 ms on the set of 10 to 15 affected transactions. It was faster than the model that had no trusted fog node by an average of 0.07 ms on the set of 30 to 35 affected transactions. This is the result of the damage sustained with the larger set which could affect a greater number of fog nodes. The second model was required to scan the graph of each affected fog node where the first model needed to only scan the global graph.

In conclusion, the system benefited from having a trusted fog node when an attack compromised the database and damage spread to more than five transactions. Further, the presence of the global graphs in the system stabilized the results and speeds the detection process, minimizing the system’s unavailability.

4.3.6. Resource Requirement Cost

Table 5 shows the space required to store the global graph file on the trusted fog node and the local graph files on each MSFN. When the number of fog nodes in the system or the number of transactions on each fog node increases, the size of the global graph file will increase. At some point it will exceed the total size of all local graph files combined. Nonetheless, the largest graph file size our experiment produced was approximately 390 kilobytes, which is still very small, and will not cause any issue in terms of space requirements, neither in the trusted fog node nor on MSFN nodes. Therefore this solution is not expensive in terms of storage requirements since hardware storage today is massive, and all graphs in our models, which represent real life situations, are sparse and will occupy an insignificant portion of storage space. Even though, those graphs exceed the worst case space requirement scenario for the adjacency list which is $E = \Theta(V^2)$, when the graph is dense and there exists an edge between all vertices $v$ of the graph to all other vertices $v'$, it will be the same space complexity as the adjacency matrix. Keep in mind, this scenario is impossible and in conflict with our submission regarding the log files since we assume the log must be serializable.
5. Conclusions

Fog computing emerged as a solution to the issues manifest in the cloud computing paradigm. Fog computing has been used successfully for smart system data management; however, fog computing is vulnerable to attackers capable of injecting malicious transactions into a fog node’s database.

We have, herein, proposed two models for smart healthcare systems that use fog computing technology to control and manage data. We consider both private and public fog computing distributions. We have primarily addressed the problem of assessing damage caused by the occurrence of a malicious attack. In both private and public systems, we developed unique algorithms to identify, then recover, all transactions affected by a malicious attack. A transaction-dependency graph is used in both schemes to monitor the activities of every transaction completed in fog computing nodes distribution. When a malicious transaction is found, the system quickly identifies and processes recovery of all other affected transactions.

We implemented the proposed models and conducted several experiments to evaluate them. we compared their performance, in terms of run time cost, considering different factors, such as the number of fog nodes, and the number of affected transaction. Through experimentation and the evaluation of the models, our distinctive models and algorithms proved viable in supporting and protecting, controlling and managing, data in smart healthcare systems. Our results show that both models are applicable and introduce only reasonable delays.

To the best of our knowledge, there are only a few works dedicated to the development of damage assessment and data recovery methods for fog computing systems. We found that all of them require scanning the entire log files of the affected fog nodes from the point of attack to the end of the logs. Continuous communication is required between all affected fog nodes on the entire system to fully detect damaged data items. Our model requires scanning only the global graph to achieve the same end; essentially shortcutting the process and therefore minimizing the time required to execute damage assessment.

As part of future work, we plan to propose a new recovery mechanism that is compatible with the damage assessment mechanism that is proposed in this paper. Additionally, we plan to extend the proposed model to applications other than the healthcare systems modeled here. We believe that our mechanisms can be modified to work in other environments, such as smart cities and industries.

Author Contributions: conceptualization, A.A. and B.P.; methodology, A.A. and B.P.; software, S.A.; validation, A.A., Brajendra Panda, and S.A.; formal analysis, A.A.; investigation, A.A.; resources, B.P.; data creation, A.A.; writing—original draft preparation, A.A.; writing—review and editing, S.A., M.A.; visualization, A.A., M.A.; supervision, B.P.; project administration, B.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.
Appendix A. Figures

Table A1. Appendix Figures Description.

| Figure | Section | Description |
|--------|---------|-------------|
| Figure A1 | Section 3.2 | Described Local Dependency Graphs for MSFN_y. |
| Figures A2–A4 | Section 4.3.1 | The impact of the different sets of affected transactions on various fog nodes number on the first model. |
| Figures A5–A7 | Section 4.3.2 | The impact of different number of transactions on various number of fog nodes on the first model. |
| Figures A8–A10 | Section 4.3.3 | The impact of different number of transactions on various number of fog nodes on the second model. |
| Figures A11–A13 | Section 4.3.4 | The impact of different number of transactions on various number of fog nodes on the second model. |

Figure A1. Described Local Dependency Graphs for MSFN_y.

Figure A2. The impact of the different sets of affected transactions on 10 fog nodes.

Figure A3. The impact of the different sets of affected transactions on 15 fog nodes.
Figure A4. The impact of the different set of affected transactions on twenty fog nodes.

Figure A5. The impact of different number of transactions on 10 fog nodes.

Figure A6. The impact of different number of transactions on 15 fog nodes.

Figure A7. The impact of different number of transactions on twenty fog nodes.
Figure A8. The impact of the different sets of affected transactions on 10 fog nodes.

Figure A9. The impact of the different sets of affected transactions on 15 fog nodes.

Figure A10. The impact of the different set of affected transactions on twenty fog nodes.

Figure A11. The impact of different number of transactions on 10 fog nodes.
Figure A12. The impact of different number of transactions on 15 fog nodes.

Figure A13. The impact of different number of transactions on twenty fog nodes.

References

1. Alazeb, A.; Panda, B. Ensuring Data Integrity in Fog Computing Based Health-Care Systems. In Proceedings of the International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage, Atlanta, GA, USA, 14–17 July 2019; Springer: Atlanta, GA, USA, 2019 [CrossRef]

2. Sanaa, K.; Haraty, R.; Masud, M. Tracking and Repairing Damaged Healthcare Databases Using the Matrix. Int. J. Distrib. Sens. Netw. 2015, 11, 914305. [CrossRef]

3. dos Anjos, J.; Gross, J.L.; Matteussi, K.J.; González, G.V.; Leithardt, V.R.; Geyer, C.F. An Algorithm to Minimize Energy Consumption and Elapsed Time for IoT Workloads in a Hybrid Architecture. Sensors 2021, 21, 2914. [CrossRef] [PubMed]

4. Panda, B.; Kazi Asharful, H. Extended data dependency approach: a robust way of rebuilding database. In Proceedings of the 2002 ACM Symposium on Applied Computing, Proceedings of the Name of the Conference, Madrid, Spain, 10–14 March 2002; pp. 446–452. [CrossRef]

5. Mukherjee, M.; Matam, R.; Shu, L.; Maglaras, L.; Ferrag, M.A.; Choudhury, N.; Kumar, V. Security and Privacy in Fog Computing: Challenges. IEEE Access 2017, 5, 19293–19304. [CrossRef]

6. Okay, F.; Ozdemir, S. A secure data aggregation protocol for fog computing based smart grids. In Proceedings of the 2018 IEEE 12th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG 2018), Doha, Qatar, 10–12 April 2018. [CrossRef]

7. Khan, S.; Parkinson, S.; Qin, Y. Fog computing security: a review of current applications and security solutions. J. Cloud Comput. 2017, 6, 1–22. [CrossRef]

8. Lin, J.; Yu, W.; Zhang, N.; Yang, X.; Zhang, H.; Zhao, W. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. IEEE Internet Things J. 2017, 4, 1125–1142. [CrossRef]

9. Wu, D.; Ansari, N. A Cooperative Computing Strategy for Blockchain-secured Fog Computing. IEEE Internet Things J. 2020, 7, 6603–6609. [CrossRef]

10. Zhu, L.; Li, M.; Zhang, Z.; Xu, C.; Zhang, R.; Du, X.; Nadra, G. Privacy-Preserving Authentication and Data Aggregation for Fog-Based Smart Grid. IEEE Commun. Mag. 2019, 57, 80–85. [CrossRef]
11. Lyu, L.; Nandakumar, K.; Rubinstein, B.; Jin, J.; Bedo, J.; Palaniswami, M. PPFA: privacy preserving fog-enabled aggregation in smart grid. IEEE Trans. Ind. Inform. 2018, 14, 3733–3744. [CrossRef]

12. Lu, R.; Heung, K.; Lashkari, A.H.; Ghorbani, A.A. A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced IoT. IEEE Access 2017, 5, 3302–3312. [CrossRef]

13. Aazam, M.; Zeadally, S.; Harras, K. Deploying fog computing in industrial internet of things and industry 4.0. IEEE Trans. Ind. Inform. 2018, 14, 4674–4682. [CrossRef]

14. Alazeb, A.; Panda, B. Maintaining Data Integrity in Fog Computing Based Critical Infrastructure Systems. In Proceedings of the 2019 International Conference on Computational Science and Computational Intelligence, Las Vegas, NV, USA, 5–7 December 2019. [CrossRef]

15. OpenFog Consortium Architecture Working Group. OpenFog Reference Architecture for Fog Computing; Budapest University of Technology and Economics: Budapest, Hungary, 2017.

16. Silva, L.A.; Leithardt, V.R.Q.; Rolim, C.O.; González, G.V.; Geyer, C.F.; Silva, J.S. PRISER: Managing notification in multiples devices with data privacy support. Sensors 2019, 19, 3098. [CrossRef]

17. Azimi, I.; Anzanpour, A.; Rahmani, A.; Pahikkala, T.; Levorato, M.; Liljeberg, P.; Dutt, N. Hich: Hierarchical fog-assisted computing architecture for healthcare iot. ACM Trans. Embed. Comput. Syst. 2017, 16, 1–20. [CrossRef]

18. Akrivopoulos, O.; Chatzigiannakis, I.; Tselios, C.; Antoniou, A. On the deployment of healthcare applications over fog computing infrastructure. In Proceedings of the 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Turin, Italy, 4–8 July 2017. [CrossRef]

19. Dastjerdi, A.; Buyya, R. Fog Computing: Helping the Internet of Things Realize Its Potential. Computer 2016, 49, 112–116. [CrossRef]

20. Vora, J.; Tanwar, S.; Tyagi, S.; Kumar, N.; Rodrigues, J. FAAL: Fog computing-based patient monitoring system for ambient assisted living. In Proceedings of the 2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom), Dalian, China, 12–15 October 2017. [CrossRef]

21. Vijayakumar, V.; Malathi, D.; Subramaniyaswamy, V.; Saravanan, P.; Logesh, R. Fog computing-based intelligent healthcare system for the detection and prevention of mosquito-borne diseases. Comput. Hum. Behav. 2019, 100, 275–285. [CrossRef]

22. Naranjo, P.; Pooranian, Z.; Shojafar, M.; Conti, M.; Buyya, R. FOCAN: A Fog-supported smart city network architecture for management of applications in the Internet of Everything environments. J. Parallel Distrib. Comput. 2019, 132, 274–283. [CrossRef]

23. Tang, B.; Chen, Z.; Hefferman, G.; Wei, T.; He, H.; Yang, Q. A hierarchical distributed fog computing architecture for big data analysis in smart cities. Proc. Asoc. Big Data Soc. 2015, 2015, 1–6. [CrossRef]

24. Amxilatis, D.; Chatzigiannakis, I.; Tselios, C.; Tsiromis, N.; Naikas, N.; Papadogeorgos, S. A smart water metering deployment based on the fog computing paradigm. Appl. Sci. 2020, 10, 1965. [CrossRef]

25. Froiz, M.; Fern, T.; Fraga-Lamas, P.; Castedo, L. Design, implementation and practical evaluation of an IoT home automation system for fog computing applications based on MQTT and ZigBee-WiFi sensor nodes. Sensors 2018, 8, 2660. [CrossRef]

26. Wang, P.; Liu, S.; Ye, F.; Chen, X. A fog-based architecture and programming model for iot applications in the smart grid. arXiv 2018, arXiv:1804.01239.

27. Zuo, Y.; Panda, B. Distributed database damage assessment paradigm. Inf. Manag. Comput. Secur. 2006, 14, 116–139. ISSN 0968-5527 [CrossRef]

28. Peng, L.; Yu, M. Damage assessment and repair in attack resilient distributed database systems. Comput. Stand. Interfaces 2011, 33, 96–107. [CrossRef]

29. Paul, A.; Sushil, J.; Peng, L. Recovery from malicious transactions. IEEE Trans. Knowl. Data Eng. 2002, 14, 1167–1185. [CrossRef]

30. Anündeya, C.; Arun K, M.; Shamik, S. A column dependency-based approach for static and dynamic recovery of databases from malicious transactions. Int. J. Inf. Secur. 2002, 9, 51–67. [CrossRef]

31. Rao, U.; Patel, D. Incorporation of application specific information for recovery in database from malicious transactions. Inf. Secur. J. Glob. Perspect. 2013, 22, 35–45. [CrossRef]

32. Haraty, R.; Sanan, K.; Zekri, A. Transaction dependency based approach for database damage assessment using a matrix. Int. J. Semant. Web Inf. Syst. 2017, 13, 74–86. [CrossRef]

33. Xie, M.; Zhu, H.; Peng, Y.; Hu, G. Tracking and repairing damaged databases using before image table. In Proceedings of the 2008 Japan-China Joint Workshop on Frontier of Computer Science and Technology, Nagasaki, Japan, 27–28 December 2007; pp. 36–41. [CrossRef]

34. Panda, B.; Alazeb, A. Securing Database Integrity in Intelligent Government Systems that Employ Fog Computing Technology. In Proceedings of the 2020 International Conference on Computing and Data Science (CDS), Stanford, CA, USA, 1–2 August 2020; IEEE: New York, NY, USA, 2020; ISBN 978-1-7281-7106-7. [CrossRef]

35. Chang, C.; Narayana Srirama, S.; Buyya, R. Indie Fog: An Efficient Fog-Computing Infrastructure for the Internet of Things. Comput. 2017, 50, 92–98. [CrossRef]

36. Kontopoulos, S.; Drakopoulos, G. A space efficient scheme for persistent graph representation. In Proceedings of the 2014 IEEE 26th International Conference on Tools with Artificial Intelligence, Limassol, Cyprus, 10–12 November 2014; pp. 299–303. [CrossRef]

37. Cormen, T.; Leiserson, C.; Rivest, R.; Stein, C. Introduction to Algorithms, 3rd ed.; MIT Press: Cambridge, MA, USA, 2009.