7-2021

Design and Development of Techniques to Ensure Integrity in Fog Computing Based Databases

Abdulwahab Fahad S. Alazeb
University of Arkansas, Fayetteville

Design and Development of Techniques to Ensure Integrity in Fog Computing Based Databases

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Engineering with a concentration in Computer Science

by

Abdulwahab Fahad S. Alazeb
King Khalid University
Bachelor of Computer Science, 2007
University of Colorado at Denver
Master of Science in Computer Science, 2014

July 2021
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council

Brajendra Panda, Ph.D.
Dissertation Director:

Susan Gauch, Ph.D.
Committee Member

Miaoqing Huang, Ph.D.
Committee Member

Paul Cronan, Ph.D.
Committee Member

ABSTRACT

The advancement of information technology in coming years will bring significant changes to the way sensitive data is processed. But the volume of generated data is rapidly growing worldwide. Technologies such as cloud computing, fog computing, and the Internet of things (IoT) will offer business service providers and consumers opportunities to obtain effective and efficient services as well as enhance their experiences and services; increased availability and higher-quality services via real-time data processing augment the potential for technology to add value to everyday experiences. This improves human life quality and easiness. As promising as these technological innovations, they are prone to security issues such as data integrity and data consistency. However, as with any computer system, these services are not without risks. There is the possibility that systems might be infiltrated by malicious transactions and, as a result, data could be corrupted, which is a cause for concern. Once an attacker damages a set of data items, the damage can spread through the database. When valid transactions read corrupted data, they can update other data items based on the value read. Given the sensitive nature of important data and the critical need to provide real-time access for decision-making, it is vital that any damage done by a malicious transaction and spread by valid transactions must be corrected immediately and accurately. In this research, we develop three different novel models for employing fog computing technology in critical systems such as healthcare, intelligent government system and critical infrastructure systems. In the first model, we present two sub-models for using fog computing in healthcare: an architecture using fog modules with heterogeneous data, and another using fog modules with homogeneous data. We propose a unique approach for

each module to assess the damage caused by malicious transactions, so that original data may be recovered and affected transactions may be identified for future investigations. In the second model, we introduced a unique model that uses fog computing in smart cities to manage utility service companies and consumer data. Then we propose a novel technique to assess damage to data caused by an attack. Thus, original data can be recovered, and a database can be returned to its consistent state as no attacking has occurred. The last model focus of designing a novel technique for an intelligent government system that uses fog computing technology to control and manage data. Unique algorithms sustaining the integrity of system data in the event of cyberattack are proposed in this segment of research. These algorithms are designed to maintain the security of systems attacked by malicious transactions or subjected to fog node data modifications. A transaction-dependency graph is implemented in this model to observe and monitor the activities of every transaction. Once an intrusion detection system detects malicious activities, the system will promptly detect all affected transactions. Then we conducted a simulation study to prove the applicability and efficacy of the proposed models. The evaluation rendered this models practicable and effective.

ACKNOWLEDGMENTS

Throughout the process of researching and writing this dissertation I have been fortunate to receive a great deal of support and guidance.

I am deeply indebted to my advisor, Professor Brajendra Panda for his unfailing attention, counsel, and guidance.

My deepest gratitude to my committee members: Prof. Susan Gauch, Prof. Paul Cronan, and Dr. Miaoqing Huang, for your time, expertise, accessibility, support, and guidance.

I am extremely grateful to Najran University and the Ministry of Education in Saudi Arabia for their financial support.

The completion of this work would not have been possible without the support of my family. I am so very thankful for my parents, my wife and kids, and my siblings and for their support, love, and grateful devotion to my continuing studies.

DEDICATION

To my parents, my siblings, my wife and my children.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

The Internet of Things is the future of the internet and the future is here. However, current infrastructure is a thing of the past and building stable and reliable infrastructure for future IoT systems necessitates consideration of the rapid growth in the number of connected IoT devices. According to estimates, by the year 2025 the number of connected IoT devices will be 75 billion [1] and produce about 79 zettabytes of data every day [2]. Further estimates indicate that by 2030 the number of connected devices will reach 125 billion [3].

The information systems currently in use cannot adequately process and transfer to the cloud the huge amount of data generated by this growth. Systems are further compromised by the limitations and restrictions on bandwidth. The rise in the number of IoT devices will create more sensitive and real-time IoT usage in connected car technologies, real-time production line monitoring, health monitoring, and video conferencing and further compromise data processing in an inadequate infrastructure. These applications need low-latency and location awareness for optimal operations [4]. The cloud and internet infrastructure and resources, in their current form, are simply incapable of managing the huge amount of data produced by this growth in the internet of things [5].

1.1 Fog Computing

To address these issues and the inadequacies of cloud computing, there is need for a new, more effective platform. Fog computing is one platform that can be used to manage the IoT. Fog computing is a virtualization architecture that handles fundamental distinguishing

services closer to the ground. Fog computing can process large amounts of data, handle storage, and networking services, and handle real-time acquisition and location awareness [6, 7]. Fog computing improves privacy and data security as data is stored and processed close to end users at the base of the network, between the devices and the cloud [8, 9].

1.2 Characteristics of Fog Computing

There are many characteristics of fog computing that are making it more popular in the field of technology, particularly in light of the IoT expansion. It is an upgrade on cloud computing with more features that are useful for new technological challenges.

Fog computing supports cloud computing and allows analytics resources to conduct intensive and extended term analytics [10]. Fog computing is close to the user, at the edge of the network thus ensuring low latency and efficient services, a characteristic critical to applications such as interactive sessions, networked games, healthcare applications, and video streaming which require low latency and location awareness.

Another indication of the benefits of fog computing is the geographical spread and high number of fog nodes. These are set to support moving vehicles and other mobile applications Fog computing provides a critical foundation for the new innovations in autonomous vehicles, ensuring quality services in connected cars technologies.

Fog computing, at the edge of the cloud and geographically spread, will prove to be beneficial in increasing bandwidth efficiency while improving data privacy. In fog computing data is processed at the fog node which means the need to send large amounts of data to the cloud is significantly reduced thus minimizing the consumption of bandwidth and maximizing the privacy of data because sensitive information will not be transmitted [6, 8, 9].

1.3 Fog Computing in Healthcare Systems

There are numerous challenges facing worldwide healthcare systems [11, 12]. Fog computing capabilities can help address these challenges. Inadequacies in current systems render the use of cloud computing technology in many current healthcare applications ineffective. For instance, the transmission of data from the sensors to the cloud and from the cloud to hospitals is slow on cloud computing. This is specifically important as healthcare systems often require urgent on-the-fly, responses that command data processing to increase efficiency. Slow data transmission compromises the efficiency of healthcare systems. [13].

Further healthcare data is sensitive and protected by the Health Insurance Portability and Accountability Act of 1996 (HIPAA) [14, 15] so the minimization of data transfer supports and guarantees data security. For this very reason, healthcare providers often prefer data storage remain within the organization. This increase in efficiency for both patient and provider is impetus for healthcare systems to adopt fog computing [16].

Fog computing is also used by some [17] to pre-process data before it is transmitted to the cloud initiating faster response to patient needs. To take advantage of fog computing, [18] introduced a novel hierarchical computing architecture for IoT-based patient monitoring systems targeted at the execution of machine-learning data analytics. [19] utilized fog computing technology to collect data traces on patient movement so that patients can access faster and more efficient services, at low latency, in the event of a medical emergency.

Fog computing is increasingly applied in the healthcare industry, attracting the attention of many healthcare technology researchers as will be discussed in Chapter 2. One of the most important issues in both fog computing and healthcare is the preservation of data

security and the privacy of end users of the system and patients. There are numerous studies that strive to address the security issues of fog computing in a healthcare setting [20], but there are still aspects of this issue that need further attention, such as assessment of data damaged by malicious attacks and determination of a method of secure data recovery after it has been damaged by malicious transactions.

1.4 Fog Computing in Critical Infrastructure Systems

Fog computing in smart city utilities, will enable the IoT and smart devices to process data faster, setting up quicker decision making and saving time. This also means the aggregation of data will be limited to the indispensable data in the cloud. The processing of huge amounts of data is required in smart cities and many countries around the world developing these cities, fog computing has compelling potential for the processing of. smart meters, traffic data, city activities, and utilities data and. Efficiently processed data will accommodate sustainable living in very developed cities [21]. However, it is important to note that it would be unrealistic to rely entirely on fog computing for processing large volumes of data. Cloud computing will necessarily continue to be used in the efforts to ensure successful smart cities.

Fog computing can be used by service providers and utility to manage and analyse consumer data efficiently, ensuring customers access to improved services. Many research studies exist on increasing the effectiveness of fog computing in smart cities and solving technical issues arising in the processing of volumes of data, especially those that require integration to the cloud [22]. IoT devices like smart meters in modern smart cities will not only generate a lot of data but the diversity of that data will have to be processed in real-

time [23]. While data in large quantity is generally very valuable, archaic technology will not harness the full value of data collected in smart cities. The current cloud is insufficient for handle such amount of data, especially tasks like processing the aggregate data, analysis and storage [24] and while data security and privacy issues have been documented by researchers, damage assessment and data recovery in the event of a cyberattack continue to require study and innovation.

1.5 Intelligent Environment Systems (Governments as example)

Governments around the world are focusing on intelligent environmental systems aimed at both conserving the environment and improving the lives of humans. Fog computing makes it possible to build these systems and optimize the benefits of environmental systems including provision of high-quality services. Still, there are risks involved in developing intelligent environmental or government systems. Perhaps the worst of those risks are data protection and data recovery in the event of a data breach.

In fact, one of the major concerns as regards government system data is the sensitivity of that data; any data breach could expose the country to attack from enemy states or terrorists. [25]. Some of the sensitive data domiciled in intelligent government systems include traffic control systems, video-conference applications, and real-time surveillance camera monitoring, and they require real-time processing and location awareness. Therefore, data damage assessment and recovery are essential in preventing data breaches as well as building a secure and dependable database. In a government data environment, the transmission of crucial data is common and needs to occur in a safe system secure from intrusion. For instance, an attack on a traffic control system or real-time surveillance camera monitor-

ing application would paralyze their vital functions like real-time processing and location awareness. The repercussions of damaged government data systems may include destruction of property or even loss of life and manipulation of economic systems. So, in addition to developing appropriate mechanisms for adaptation of fog computing in intelligent systems, developing adequate security features capable of responding to attacks by providing fast and accurate damage assessment and recovery techniques is of cardinal importance.

1.6 Computing System Security

Securing a computing system is critical in every situation and industry. There are three key phases required to ensure a system is protected and secure. The first phase is data protection. Techniques like access control, encryption, auditing, and authentication are applied to the system to secure and protect the data. The second phase is an intrusion and compromise detection system This can be software or a device that monitors the system with the goal of sensing any malicious activities or policy violations and raising an alarm so that it can be addressed. The most vital component of this phase is timely detection and notification of the system compromise [26]. This phase uses an Intrusion Detection System (IDS) which can observe system activity and alerts the person who is in charge of any unusual activity. This requires more assistance from the third phase, which includes damage assessment and data recovery ensuring the integrity and accessibility of the data in the system.

The third phase is critical in detecting any further transgressions on the system and making sure that the database has been restored and is secure. This phase can be divided into two important stages: damage assessment and data recovery. Damage assessment is

the primary process in this phase and is used to identify compromised transactions and data items. Data recovery restores the damaged data to its last consistent state before the attack. To make this possible, log files must retain information about the changes applied to data items by various transactions in the system [27].

Delay time is computed in the damage assessment stage from the moment the IDS identifies a malicious transaction until all transactions or data items affected by the introduction of the malicious transaction have been identified. During this time, the system will be unavailable so, we aim to minimize the this process requires as much as possible. Time latency for the data recovery stage is computed from the moment all affected transactions have been received until all data items have been recovered. During this time only the affected data items will be unavailable for use so as before, we aim to minimize the time required for this process.

The system is not capable of assessing all damaged data through the damage assessment and data recovery algorithms with the use of log files on each fog node independently in a fog computing environment. For instance, as shown in Fig. 1.1, if the Intrusion Detection System (IDS) identifies T_3 in Fog_1 and T_8 in Fog_2 as malicious transactions and if every involved fog node individually tests their local log files, the system will then just detect T_5 and T_6 as affected transactions in Fog_1 and T_{10} as an affected transaction in Fog_2. However, T_7 and T_8 in Fog_3, dependent on the affected transactions in Fog_1, and T_9 in Fog_2, dependent on the affected transactions in Fog_3, have been affected and will not be detected by these independent assessments. So to confirm that all affected transactions in the system are detected there must be collaboration between fog nodes.

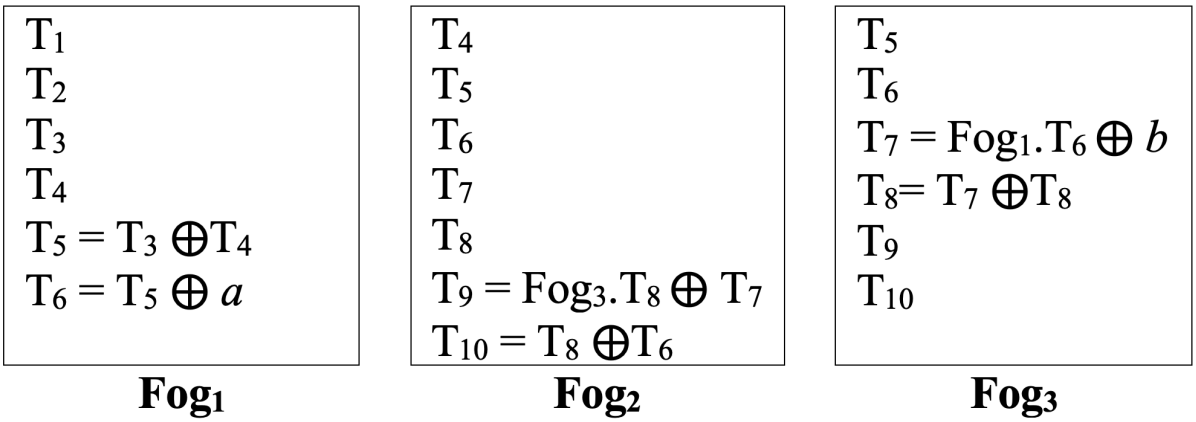

Figure 1.1: An example of the necessity of having cooperation between fog nodes to obtain accurate data damage assessment. $\oplus$ notation could indicate any possible operation.

1.7 Summary of Contributions

Securing any computing system is necessary for the protection and security of the system and its data. To that end, three primary phases are essential to ensure a system is protected and secure [27]. The first phase in data protection utilizes methods including access control, auditing, authentication, and encryption. The second phase focuses on intrusion detection and utilizes software or a device that observes the system with the intent of detecting any malicious activity or policy violations. If intrusion of the system occurs, the third phase, which includes damage assessment and data recovery, ensures the integrity and availability of system data. This third phase is essential in detecting any additional data corruption and in ensuring the system is returned to a secure state.

The main objective of this research is to demonstrate the ability of the proposed algorithms to detect data damaged in an attack in systems maintaining extremely sensitive databases and employing fog computing environments. Critical infrastructure systems, smart cities, intelligent government systems, and healthcare systems are among those where

databases store sensitive information. They are prime targets for attacks.

The first goal of this research is the construct of two models for fog computing based on healthcare systems. The first architecture will use fog modules with heterogeneous data, and the second will use fog modules with homogeneous data. A unique approach to damage assessment will be proposed for each module.

Traditional damage assessment and data recovery algorithms usually delete data affected by an attack to guarantee the integrity of the database. In the proposed algorithms the affected data will be identified and retained for use in any future investigation. Suppose, for instance, a critical patient information system was attacked. A treating physician, unaware of the data violation, administers medication to a patient with life-threatening allergies. Identifying, and retaining the original, affected data has the potential to avoid further consequences.

The second goal is to introduce a unique model that uses fog computing to manage utility service companies and consumer data in the infrastructure systems of smart cities. A novel technique to detect and assess data items that are affected by a malignant attack will be proposed. This proposal will generate a method for recovery of the original data and a strategy for returning the database to a state consistent with that prior to the transgression. The construct of a damage audit table, a structure, that will be used to collect data needed in the recovery process, will be implemented.

The third goal of this research is to design a novel model for an intelligent government system that will use fog computing technology to control and manage the data in the entire system. Unique algorithms that will sustain the integrity of data in the system in the event of a cyberattack, are proposed for this segment of the research. These algorithms are intended

to maintain the security of systems experiencing the execution of a malicious transaction or modified data in the database of fog node. A transaction dependency graph will be implemented in this model to observe and monitor all transactions and quickly detect all affected transactions if a malicious transaction is found.

2 Literature Review

2.1 Fog Computing

The benefits of cloud computing are synonymous to its advantages especially to businesses and other related entities. One such benefit revolves around the reduction of workload as well as the administrative burden of system maintenance and data management. Work is made easier, cheaper, and fast with cloud computing. The need for low latency, mobility support, and geo-distribution and location awareness must be addressed with the changing needs of database management [6]. This development pushed for the implementation of a new infrastructure by Cisco Systems known as fog computing in 2014. Bonomi et al. [28] advised that fog computing is just an addition or an enhancement on cloud computing technologies that are central to edge computing technologies to handle issues facing the computing system such as high latency and inflexibility.

Fog computing as a computing infrastructure has elicited the attention of the academia and other interests such as for industrial research [29, 30, 31, 32]. Several researchers developed architecture models for the fog system as discussed in Section [33, 34] in general survey or study of the challenges, issues, and future direction.

2.1.1 Fog Computing Features

Fog computing has been known to advance the features of cloud computing to improve database operations. The first feature associated with fog computing concerns location,

which can be termed as the edge of networks. This feature enhances the quality of services provided that are high and reduces the latency from high to low, a factor that helps save time and system performances. Application areas include the healthcare industry through the monitoring applications used for patients, gaming, and online streaming of videos. Mobility is a substantial feature in fog computing that is promoted through the broad dispersion of fog nodes, hence enhancing its geographical accessibility. An example of the applicability of these characteristics is in moving vehicle services. This application makes fog a vital keystone in the provision of high-quality services for associated car technologies. Fog's feature of being located at the edge of the cloud and its widespread geographical distribution help increase bandwidth efficiency, privacy, and security of sensitive data. Most of the local data in a database system are processed by fog nodes, which indicates that a reduction in the amount of data sent to the cloud for processing will help reduce the consumption of bandwidth. This activity ensures the maximization of privacy related to sensitive data transmission [6, 8]. Therefore, fog computing should work as a proper, appropriate platform for numerous sensitive Internet of Things (IoT) services and applications, which could include connected vehicles, electricity automation systems, and smart cities.

In their study regarding the present direction of patterns for technology usage and the development of enabling technologies, Vaquero et al. [35] proposed a comprehensive definition of fog computing, that is, "a scenario in which a large number of heterogeneous ubiquitous and decentralized devices communicate and there exists potential cooperation among themselves and with the network to complete storage and processing tasks without third party intervention. They further provided an explanation for why a subscribed communication model is required when data only requires to be sent or published. This helps

reduce traffic in the network, keep congestion problems at the edge of the network, and have a positive influence on privacy protection."

The importance of preprocessing whereby data gets changed and the movement of data to the cloud utilizing computing technologies such as smart gateway was discussed by Aazam et al. [17]. They devised a new infrastructure for smart gateway, which was mainly concerned with smart homes and would enable the connection to IoT devices. They also proposed a new architecture for a smart gateway using fog computing. Ivan et al. [36] investigated the merits of fog computing for services in various dimensions. The services include electricity networks, IoT, CPS, and automation of buildings. They studied an updated model for fog computing and its security issues.

2.1.2 Fog Computing Security and Privacy Issues

Cloud computing may not help resolve security and privacy issues, such as data protection, data availability, authentication, and user communication, leaving the role to fog computing [37]. The combination of crucial data enhances privacy and security because most of the data are processed locally at the edge of the network [38]. 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 [39]. Several security and privacy issues can be mitigated by integrating fog computing with IoT infrastructure [40].

However, security and privacy issues have largely contributed to the research and contributions of novel concepts and improvement solutions [37, 41, 42]. 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 [43, 44, 45, 46, 36, 47, 48].

A fog computing architecture was mentioned in [49], 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 [44]. 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.

Lyu et al. also addressed the element of smart cities and smart meter readings [45] 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.

Mohammed et al. [50], provided an encryption-based scheme to avert rogue fog computing nodes and stop them from risking end-user data security while upholding reduced time latency and communication overhead between the cloud and the fog nodes. They also proposed a blockchain-encryption-based scheme integrating the CP-ABE algorithm with blockchain technology for the detection of rogue fog nodes federated with other fog nodes. When rogue fog nodes are detected, they are ousted, making them incapable of accessing data encrypted with the fog federation's attributes; data stored in fog nodes in the same federation or in the cloud are made safe from the rogue. They also relied on blockchain technology to perform authorization in a distributed manner and track the encrypted data through fog federations [51].

2.1.3 Fog Computing Architecture in Critical Infrastructure and Intelligent System

Fog computing has been widely applied in the healthcare industry, which has attracted the attention of numerous researchers [11, 18]. Azimi et al.[18] 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. [11] 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. [52] devised a new structure of using fog computing to monitor patients for ambient assisted living.

Vijayakumar et al. [53] 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. [54] 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. [55] 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. [56] 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. [46] 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. [57] 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 [58] 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.

2.2 Database Security

A computing system needs to be secured to guarantee the protection and security of the system and its data. Database security does not only comprise the technical aspects but also moral and ethical as well as legal issues, in which specific laws exist to help regulate information disclosure according to Pernul [59]. The three principles of database security can be described as secrecy, integrity, and availability of stored data. To achieve this objective, three primary phases are essential to ensure that a system is protected and secure [27]. The first phase applied in the protection of data relies on methods such as access control, auditing, authentication, and encryption. This first phase involves data that are at rest and not moving. Security is provided for such data including limiting access at entry and endpoints. Access control of data also involves password management or classification of sensitive data.

The second phase relevant to the protection of data emphasizes the detection of intrusion and relies on devices that assess the system to detect any malicious activity or policy violations. Forms of intrusion include misuse of authority such as theft of media or modification of data, logical inference and aggregation that concerns the sensitivity of data, masquerade that involves unauthorized access by an intruder masquerading to be an authorized user, or even bypassing of controls such as passwords, browsing, or through Trojan horses according to Pernul [59]. This second phase involves data that are in transit. In this phase, the security of the system borders as well the monitoring of the movement of data and identifying threats are ensured.

In the case of system intrusion, the third phase, which includes damage assessment

and data recovery, safeguards the integrity and availability of the system data. This third stage is critical in the detection of any additional data corruption and ensuring that the system reverts to a secure state. This phase involves data that are already being utilized. Detection encompasses user monitoring for those who have access to sensitive data.

This research objectively demonstrates the ability of the proposed algorithms in the detection of system data that are damaged in an attack while maintaining very sensitive databases and applying fog computing environments. Critical infrastructure systems, smart cities, intelligent government systems, and healthcare systems are vulnerable to attacks because of the sensitivity of the information stored in their databases.

2.2.1 Damage Assessment and Data Recovery

Research has studied damage assessment and data recovery in traditional databases. Models and mechanisms for data recovery following cyberattacks have been proposed [60, 61, 62, 63, 64]. However, academic contribution to damage assessment and data recovery in modern database infrastructure, such as fog and edge computing, appears lacking. Damage is usually caused by elements such as computer viruses, Trojan horses, logic bombs, or trap doors. Recovery can be termed as 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 ensure the accuracy of the algorithms on databases in question. Kaddoura et al. [65] recommended the use of a single matrix for damage assessment and recovery of algorithms. Other techniques such as parsing of the database log to check for the affected transactions were recommended by [66]. Damage assessment usually takes place after the set preventive measures fail to prevent malicious

attacks. Malicious attacks generally intend to damage data stored on the database system. Damage can occur to the whole system of specific objects that are difficult to detect according to Liu at el. [67].

Panda at el. [60] recommended the data dependency method instead of transaction dependency. Every read and write operation of a transaction must be classified into one of five different types grounded on the data dependency between each operation. A directed graph, whose function is characterized by offering up data items that have been affected in the database, is utilized.

The column dependency-based approach presented by Chakraborty et al. [68] deduces the relationship between transactions to determine which transactions affected by malicious attacks need to be recovered. In this approach, the recovery of data after attacks, which is usually time consuming, takes 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.

Liu and Yu [67] 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 on the system through the

generation of a coordinator for any cleaning transaction.

Panda et al. [69] used the data dependency-based approach to assess the damage that could occur from electronic attacks and then return the database to a consistent state. They introduced two algorithms. In the first one, damage assessment and recovery algorithms are effected simultaneously, resulting in considerable delays due to blocking the system until the whole procedure is complete. The second algorithm handles this because the system will be soon available after all the affected and damaged data are identified and blocked. Fu et al. [61] introduced new dependencies that relied on analyzing inter transaction dependency relationships to resolve damage assessment. They proposed and evaluated four different dependency relationships between transactions that could transfer the damage.

Ammann et al. [66] also introduced algorithm sets and recommended a mechanism that would only work on the damaged portions of the database to restore the log files immediately when proposed damage is assessed and perform data recovery algorithms. These algorithms can 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 and how the process provides data for the repair of damaged transactions.

In [63], 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 point or 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 claimed that this approach can trace the data as they spread through different machines. The BI Tables are utilized to

repair the damaged data without even assessing the log file.

Zuo and Panda [70] consequently introduced two dissimilar methods for the detection of affected transactions in the distributed database system. The first method utilized the peer-to-peer model, which is much useful when assessing a single point of failure, rather than multiple failure points in the system. The second method is a centralized model whose efficiency is high in the case of a large-scale distributed database system as a result of the minimization of network communications among the sites.

Haraty et al. [65] proposed an algorithm that would track 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.

Additionally, Sobhan and Panda [71] recommended a new logging protocol that records all the necessary information for the full repair of a database that is updated by committed but affected transactions. Lala and Panda [72] devised a damage assessment model as well as four associated data structures to hasten accurate data recovery. These rely on dependency relationships among the transactions, which in turn update the database. Similar sentiments or recommendations were echoed by Panda and Zhou [73] to devise accurate, fast methods for damage assessment. Two approaches, which include the use of transaction dependency relationships to determine the affected transactions, were recommended in the attacked database. The other second approach considers data dependency relationships to identify the data items affected by the attack for future use in terms of recovery.

The most targeted sites for attacks are those that contain sensitive information such as E-government sites. In the case of a malicious attacks, such sites require fast damage assessment as well as recovery of data. Kurra et al. [74] recommended a model that could advance damage assessment and recovery through minimal log access using multiple agents, which helps save time.

Rao and Patel in [62] introduced a methodology for data recovery based on the inclusion of application specific metadata to form transaction dependencies. For the best performance, a column-based transaction dependency is used in this method. Their model restores only affected transactions and skips malicious transactions and valid transactions.

2.2.2 Data Dependency versus Transaction Dependency

A collection of operations or tasks, such as reading and writing, can be termed as a transaction. Every operation has limits, such as minimum processing units that cannot be further divided. Whenever a data item is written by a transaction, it is possible for it to be read by another transaction. This situation creates a dependency between those two transactions. For example, transaction T_x reads data item D_1 and then writes data item D_2. Similarly, T_y reads item D_2, making transaction T_y dependent on T_x. Owing to this form of dependency, if T_x becomes affected by a malicious attack, T_y will also be affected. Thus, in a fog computing environment, a data item that is informed by any transaction in a fog node can be read by any other.

Therefore, damage assessment relies on either data dependency or transaction dependency. Data dependency assesses if data items in the database are written after reading data items that are affected by an intrusion. By contrast, transaction dependency situates

a transaction that is most likely to be affected if a data item that is to be read is written by a malicious of an affected transaction according to Haraty et al. [65]. In the study of Panda and Yalamanchili [75], transaction dependency focuses on malicious transactions for system recovery. Transactions affected by malicious transactions are also placed in line for recovery. These transactions that are malicious of those that are dependent on them are usually undone. Others that are affected by malicious transactions are redone. Untouched transactions have nothing done to them because they are neither dependent on malicious transactions nor affected by them. Transaction dependency is a self-healing system according to Xia et al. [76]. Consequently, data dependency has all the data items affected by malicious transactions returned to their previous state that could be the values of the data. Therefore, data dependency methods for database recovery undo and redo the affected operations or rather transactions, and do not undo all the transactions according to Zheng [77].

2.2.3 Flushing the Data

Fog computing is known to have storage limitations, and this calls for the periodic flushing and removal of data and corresponding log files of fog nodes and their permanent storage in the cloud [29, 78]. Each fog node, therefore, will have efficient and automated access to its own cloud space. In specific cases, further assessment of the flushed data needs to be undertaken using the proposed algorithm. However, assumptions are made that this action will be executed at optimal intervals when the proposed approach has been launched and all affected transactions detected or the IDS has provided clearance to the flushed data.

The advantages of this [29] are its capability to enhance the runtime efficiency of the proposed algorithm because the size of the fog database used in our approach is likely

to be diminished. Another advantage is the improved efficacy in detecting affected transactions among all data that is flushed to the cloud because all the data will be in one high-performance machine. Flushing is, however, meant to be performed when the database is in a secure state and the commitment of all transaction is complete.

3 Ensuring Data Integrity in Fog Computing Based Healthcare Systems Model

Some of the following paragraphs, figures, and algorithms that will be introduced in this chapter have been already published in our work [43] as shown in the publications and reprint permissions chapter 7.

3.1 Introduction

The future of the internet will be in the Internet of things (IoT), which is evidenced by the significant increase in wearable technology, smart homes and buildings, connected vehicles, and smart grids. The estimated number of connected IoT devices in 2030 is nearly 125 billion, which will produce an enormous amount of data [3]. Due to the limitation and restriction of bandwidth, as well as the rapid growth in the amount of data produced, the current information system architecture will be inadequate for managing and moving that volume of data to the cloud. In many scenarios, it could be impractical to do so, especially with the increasing number of IoT devices in use. Additionally, our current society has incorporated a lot of sensitive and real-time applications of IoT as integral parts of our lives for instance, through the use of connected car technologies, video conference applications, health monitoring, and real-time production line monitoring, all applications requiring low-latency and location awareness in order to provide satiable and high-quality services [4].

The need for a new platform will become necessary to address the above-mentioned issues. For that purpose, fog computing, introduced by Cisco, is a virtualization architec-

ture that provides many fundamental distinguishing services close to the ground, including the ability to process copious amounts of data, storage, and networking services, making fog computing especially appropriate for many sensitive applications that require real-time acquisition and location awareness [6]. It enhances privacy and security because the data is kept and computed close to end users at the edge of the network, between the end devices and a cloud [8].

Fog computing has several unique characteristics that will not only establish it as an extension of the cloud, but will also provide extra privileges over the cloud. The first feature is its location at the edge of networks, providing end users with high-quality services and low latency. Many current applications require location awareness and low latency to provide a higher quality of services and performances, such as healthcare applications, networked games, video streaming, and interactive sessions. Another essential characteristic is the widely dispersed and significant numbers of fog nodes that will be geographically available, a design that supports mobility in many applications, including service in moving vehicles. This will make fog an important cornerstone of providing high-quality services for connected car technologies. Both fog's location at the edge of the cloud and its geographically widespread distribution will also contribute to the benefits of increasing bandwidth efficiency and enhancing the privacy and security of sensitive data. Most of the data will be processed locally at the fog node, meaning that the amount of data needing to be sent to the cloud for processing will be diminished, helping to minimize bandwidth consumption and maximize the privacy of transmitting sensitive data [6, 8].

The healthcare system faces several challenges [11, 12] that some features of fog computing mentioned above may be able to solve. Therefore, the healthcare system can

capitalize on fog computing to create better experiences and services for both patients and providers. One of the most crucial issues in both the fields of fog computing and healthcare is preserving the security and privacy of consumer or patient data. While several studies have sought to solve the security issues of fog computing in a healthcare environment [20], there are still aspects of this issue that should be given further attention, such as assessment of the damage data could suffer from malicious attacks and determination of how to securely recover data from malicious transactions. Damage assessment and data recovery are essential in creating secure and reliable databases, particularly for the transmission of sensitive data, such as that of the healthcare environment. For example, if an intrusion detection system (IDS) detects malicious transactions in the system, any other transactions that read this data will also be affected, resulting in any doctor's decision made based on affected transactions, potentially putting a patient in danger of harm. To the best of our knowledge, there has been no previous work done on the development of damage assessment and data recovery methods for fog computing systems. We present two models for using fog computing systems that manage healthcare data: architecture using fog modules with heterogeneous data, and a second architecture using fog modules with homogeneous data, using unique approaches for each module for assessing the damage caused by malicious transactions, for accurately recovering data, and for identifying affected transactions for future investigation.

3.2 The Models

In this section, we introduce two possible architectures for using fog computing to manage healthcare data. For each architecture, we propose suitable algorithms to determine the effect of an attack on the system and identify data damaged either directly or indirectly

so that they can be recovered quickly. We assume in both cases that the intrusion detection system (IDS) is responsible for detecting the malicious transactions in the system and will provide a list of these transactions. In addition, each fog node in both architectures must have its own log file. Both architectures proposed here also use a strict serializable history, and the log files cannot be modified by users at any time.

3.3 Model Notations

Table 3.1 shows the description of notation to be used in our proposed approaches.

Table 3.1: Notation used in our proposed approaches description.

Notation	Description
FDR	The main fog node, which is accessed by the health care providers to read and write about the patients.
FM	Fog node for specific monitors used to collect the data from the patients using the IoT devices, e.g. Fog of Heart Monitors.
MT_L	List of detected malicious transactions done by IDS.
Aff_Lfdr	List of all affected transactions that have been detected in FDR.
Aff_Lfdr$_j$	List of all affected transactions that have been read by FDR from FM$_j$.
Aff_Lfm$_j$	List of all affected transactions that are detected by the proposed algorithm in the fog node FM$_j$.
Aff_Lfdr$_{cloud}$	List of all affected transactions that done by cloud over the flushed data of FDR fog node.
Aff_Lfdr$_{cloud,j}$	List of all affected transactions that done by cloud over the flushed data of fog node j and have been read by FDR node.
Aff_L$_{cloud,j}$	List of all affected transactions that done by cloud over the flushed data of fog node j.

3.4 The First Proposed Architecture: Fog Nodes with Homogeneous Data

In this model, data will be written to one fog node, which can be read by multiple fog nodes. The patient data will be collected using end-users and Internet of things (IoT) devices, such as sensors, and sent to the proper fog node, which we call fog monitors (FM). For example, information about a patient's heart rate will be sent to the Fog of Heart Monitors, and his or her blood pressure reading will be sent to the fog of blood pressure monitors. Therefore, each fog node FM will contain only homogeneous data.

Additionally, in this model there is a main fog node, known as the Fog for Doctor Reports (FDR), which has access to read any necessary data from the other fog nodes (FMs). Healthcare providers, however, will only have access to the main node. When a doctor wants to check patient records, he or she must read the patient's data from the FDR and write reports, concerns, and prescriptions based on that data, as shown in Fig. 3.1. We assume that each fog node will have the ability to perform some basic operations with the data, such as calculating a patient's average body temperature over a certain time frame or aggregating the totals of selected data values.

3.4.1 Damage Assessment Algorithm for the First Model.

Once the IDS detects all malicious transactions in the system, it will send to each fog node a list of malicious transactions that have been detected on it. Let us say that a malicious transaction T_i has been detected by the IDS on Fog_1. Fog_1 will then perform the following procedures:

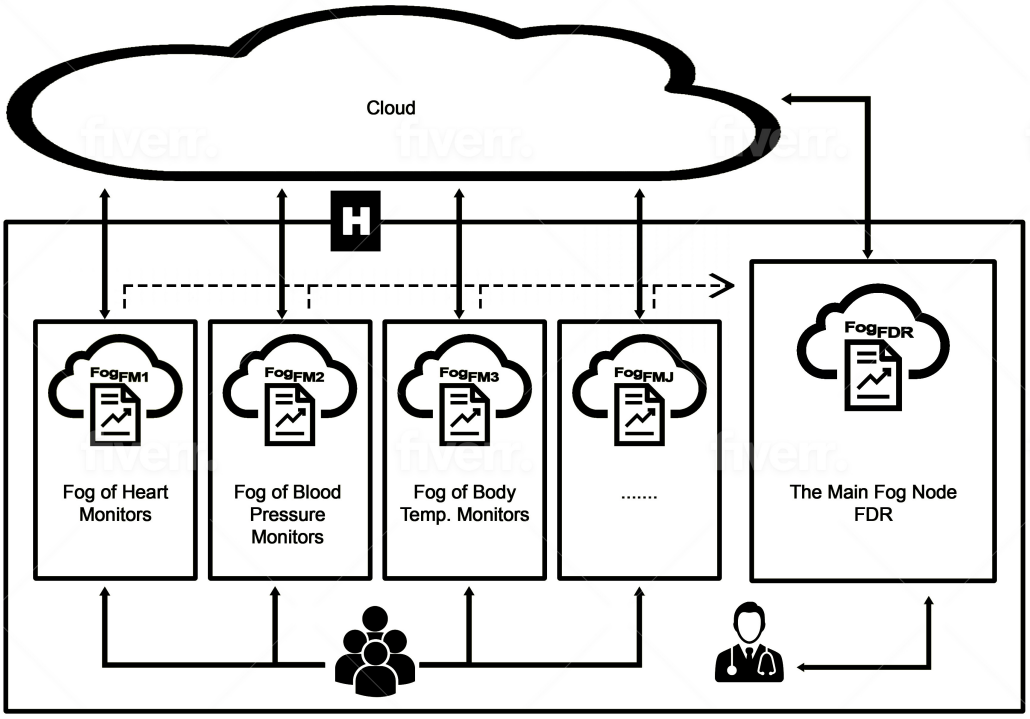

Figure 3.1: First proposed architecture "fog nodes with homogeneous data".

3.4.1.1 First procedure:

Fog_1 will check its local log to confirm whether T_i is still in its local database (DB).

- If T_i is not in the DB, it has been flushed to the permanent storage in the cloud.

  In this case, the list of malicious transactions will be forwarded to the cloud to identify all affected transactions by checking data received from all fog nodes. (This step will be more effective and efficient via cloud computing, since all data is now in one high-performance machine that saves time by reducing unnecessary communication between the fog nodes.) After the cloud receives those lists, it will scan the flushed logs to identify any affected transaction that is dependent on any of the damaged transactions from the list.

Once the cloud finishes the damage assessment procedure and identifies all affected transactions, it will create a list of affected transactions Aff_L$_{cloud,j}$ for each fog node in the system which has affected transaction in its flushed data. The Aff_L$_{cloud,j}$ lists will be sent to their corresponding fog nodes to be used as input for our proposed algorithms to do further detection. The cloud will identify all affected transactions read by the FDR, since the FDR log file is flushed to the cloud along with the data.

- If T$_i$ is still in the local DB of Fog$_1$, then the second procedure will be performed, as outlined below, to detect any additional affected transactions.

3.4.1.2 Second procedure:

Three algorithms will be launched once the fog receives the list of malicious transactions MT_L, detected by the IDS, or Aff_L$_{cloud,j}$ that is sent from the cloud. The first algorithm, Algorithm 3.1, allows FM fog nodes to further detect all damaged transactions and mark them as affected. This algorithm will also confirm whether the main fog node (FDR) has read any of these identified transactions. If so, the algorithm will send a list Aff_Lfdr$_j$ to the FDR to use as input for further detection. The other two algorithms, Algorithms 3.2 and 3.3, are for the main fog node FDR. The Algorithm 3.2 could run simultaneously with Algorithm 3.1 once MT_L, Aff_Lfdr$_{cloud}$,or Aff_Lfdr$_{cloud,j}$ list is received, to ensure fast detection of all affected transactions. The FDR fog node will not launch Algorithm 3.3 until any list of affected transactions from the FM fog nodes is received.

To illustrate the first model, suppose the IDS system detects T$_i$, Alice's body temperature, as a malicious transaction that is collected by the end user S$_{j1}$. S$_{j1}$ is connected

Algorithm 3.1 FM Fog Nodes Assessment Algorithm

Input:

- List of detected malicious transactions MT_L done by IDS, or Aff_L$_{cloud,j}$ list from the cloud.
- The local FM log file.

Output:

- List of all affected transactions Aff_Lfm$_j$ that will be detected by our proposed algorithm in the Fog node FM$_j$.
- Aff_Lfdr$_j$ List of all affected transactions that have been read by FDR.

The Algorithm:

1: Creates a new affected-transactions list Aff_Lfm$_j$ and initializes to null where j is the current fog node ID.
2: Creates a new affected list Aff_Lfdr$_j$ and initializes to null.
3: Takes MT_L / Aff_L$_{cloud,j}$ as input
4: Copies all malicious / affected transactions T$_i$ that exist in FM$_j$ from MT_L / Aff_L$_{cloud,j}$ list to Aff_Lfm$_j$ list
5: **if** Aff_Lfm$_j \neq \emptyset$ **then**
6: Scan the local log and
7: **for** every T$_k \in$ FM$_j$ that is dependent on any T$_i \in$ Aff_Lfm$_j$ **do**
8: Mark T$_K$ as affected transaction
9: Add T$_K$ to the Aff_Lfm$_j$ list
10: **end for**
11: **for** every T$_i \in$ Aff_Lfm$_j$ **do**
12: Check if T$_i$ has been read by FDR
13: Add T$_i$ to the Aff_Lfdr$_j$ list
14: **end for**
15: **end if**
16: Send Aff_Lfdr$_j$ to FDR to do further detection
17: Send Aff_Lfm$_j$ for data recovery.

Algorithm 3.2 FDR Fog Node Assessment Algorithm "1"

Input:

- List of detected malicious transactions MT_L, Aff_Lfdr$_{cloud,j}$, or Aff_Lfdr$_{cloud}$.
- The local FDR log file.

Output:

- List of all affected transactions Aff_Lfdr.

The Algorithm:

Creates a new affected-transactions list Aff_Lfdr and initializes to null
Takes MT_L / Aff_Lfdr$_{cloud,j}$ / Aff_Lfdr$_{cloud}$ as input
Copies all malicious / affected transactions T_i that exist in FDR from the input list to Aff_Lfdr list
if Aff_Lfdr $\neq \emptyset$ **then**
  Scan the local log and
  for every $T_k \in$ FDR that is dependent on any $T_i \in$ Aff_Lfdr **do**
    Mark T_K as affected transaction
    Add T_K to the Aff_Lfdr list
  end for
end if
Send Aff_Lfdr for data recovery.

Algorithm 3.3 FDR Fog Node Assessment Algorithm "2"

Input:

- Aff_Lfdr$_j$ list of affected transactions received from other fog node FM_j.
- The local FDR log file.

Output:

- List of all affected transactions Aff_Lfdr.

The Algorithm:

for each a new affected list that is received from FM_j **do**
  Creates a new affected-transactions list Aff_Lfdr and initializes to null
  Takes Aff_Lfdr$_j$ as input
  if Aff_Lfdr$_j \neq \emptyset$ **then**
    Scan the local log and
    for every $T_k \in$ FDR that is dependent on any $T_i \in$ Aff_Lfdr$_j$ **do**
      Mark T_K as affected transaction
      Add T_K to both list the Aff_Lfdr and Aff_Lfdr$_j$
    end for there is no affected transaction has been read from FM_j
  end if
  Send Aff_Lfdr for data recovery.
end for

to the fog node (FM_j). Now assume that FM_j has performed some local operations on its data before the detection, as a matter of general routine, including T_i, such as:

1. Aggregate all the patients' body temperature readings over a certain time $T_{agg,j}$
2. Calculate Alice's average body temperature $T_{avg,j}$ over the last six hours.

If FDR reads T_i, $T_{agg,j}$, and $T_{avg,j}$ from FM_j, then any transaction $T_j \in$ FDR that is dependent on T_i , $T_{agg,j}$, or $T_{avg,j}$ will be affected. Note that there is no way FDR can know that $T_{agg,j}$ and $T_{avg,j}$ are dependent on T_i, since they were calculated locally on the FM_j node. Therefore, any transactions that belong to FDR and are dependent on $T_{agg,j}$ or $T_{avg,j}$ will not be detected by FDR itself until FDR gets a list of all affected transactions from FM_j. Assume, for the following transactions $T1_j$, $T2_j$, and $T3_j \in$ FDR, that:

- $T1_j$ is the doctor's report, that is dependent on the malicious transaction T_i;
- $T2_j$ is dependent on $T_{agg,j}$ to do any kind of study or to make a budget of the hospital;
- $T3_j$, the prescription given to Alice, is dependent on $T_{avg,j}$ and has some side effects that may affect other data, such as heart rate or blood pressure.

Now suppose the IDS has already detected all malicious transactions including T_i and sent them as list MT_L to the system. Then the first procedure will check each malicious transaction on MT_L to determine whether it has been flushed to the cloud. Suppose, however, that T_i is still on the local DB of FM_j. In that case, the second procedure will pursue further detection.

Thus, the FDR will run Algorithm 3.2 while all other affected fog nodes FM_j run Algorithm 3.1 simultaneously. As Algorithm 3.1 runs, each affected FM fog node will create

two lists:

- Aff_Lfm$_j$ will get a copy of all malicious transactions that exist in FM$_j$ from MT_L list. All detected damage transactions in FM$_j$ will be added to this list.

- Aff_Lfdr$_j$ which will contain all affected transactions read by FDR and detected by Algorithm 3.1 in FM$_j$.

In lines 6 - 10, the algorithm will go through every transaction in the log of FM$_j$ to confirm whether any of them are dependent on malicious or affected transactions from the list Aff_Lfm$_j$. If so, these transactions will also be marked as affected and added to Aff_Lfm$_j$. Thus, T$_{agg,j}$ and T$_{avg,j}$ will be added to Aff_Lfm$_j$, since they are both dependent on the malicious transaction T$_i$.

In lines 11 - 14, this loop of algorithm is to check only the final Aff_Lfm$_j$ list of all malicious and affected transactions and confirm whether any have been read by FDR. If so, these will be added to Aff_Lfdr$_j$. Therefore, T$_i$, T$_{agg,j}$, and T$_{avg,j}$ will be added to Aff_Lfdr$_j$, since they have been read by FDR. Aff_Lfdr$_j$ will then be sent to FDR to use as input for Algorithm 3.3 to do further detection, while Aff_Lfm$_j$ will be sent for data recovery.

Once Algorithm 3.3 receives Aff_Lfdr$_j$, it will create a new affected list, Aff_Lfdr. The algorithm will go through all transactions in the FDR log to determine whether any depend on the malicious or affected transactions from Aff_Lfdr$_j$. If so, these will be marked as affected transactions and added to both Aff_Lfdr and Aff_Lfdr$_j$. T1$_j$, T2$_j$, and T3$_j$ will thus be detected by the end of this loop, and all three will be added to both lists. Then Aff_Lfdr will be sent for data recovery.

3.5 The Second Proposed Architecture: Fog Nodes with Heterogeneous Data

In this model, we assume that all fog nodes have the same capability as well as the ability to perform basic operations on data. However, in the "fog nodes with heterogeneous data" model, there is no main fog node; each hospital department will have its own fog node. Thus, when a patient is moved from one department to another, his or her data will also be moved to the fog node of that department, as shown in Fig. 3.2. The healthcare providers in each department will have only access to patient data through their own fog node and will write records or reports based on this information. All patients' data will be collected using end users and IoT devices, such as sensors, and will be sent to the local fog node FM in the same department.

This model can be applied outside of hospitals as well, since patient data in smart cities will likewise move from one fog node to another while patients are traveling. Write operations can also be accessed from all nodes. Thus, unlike the "fog nodes with homogeneous data" model, malicious transactions in the "fog nodes with heterogeneous data" model could spread quickly and cause severe data damage. In order to resolve this issue, the fog nodes need to cooperate with each other.

3.5.1 Damage Assessment Algorithm for the Second Model.

3.5.1.1 First procedure:

As in the first model, when malicious transactions are detected by the IDS, the affected fog node will scan for the presence of that transaction in its local DB. If the transaction is not there, that means it has been flushed to the cloud. The full procedure in such a case

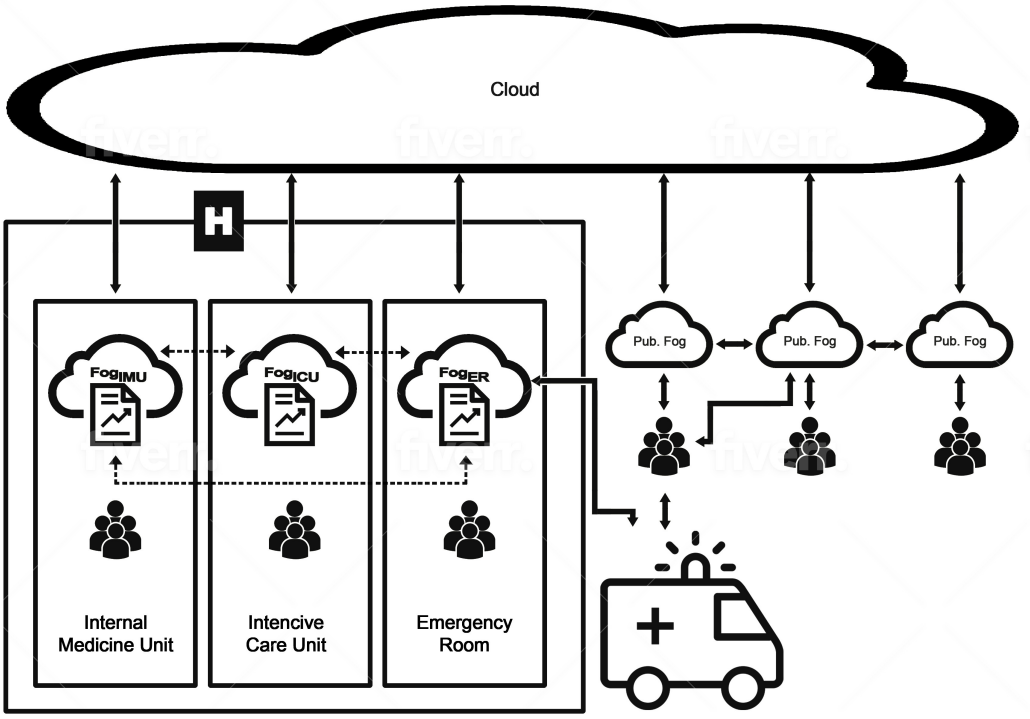

Figure 3.2: Second proposed architecture "fog nodes with heterogeneous data"

is similar to that explained in section 3.4.1.1.

3.5.1.2 Second procedure:

Our proposed Algorithm 3.4 will be performed in case one of following list is received:

- The list of malicious transactions MT_L.
- The list of affected transactions from another Fog node Aff_Lfmx$_j$.
- The list of affected transactions that has been detected by cloud in the flushed data Aff_L$_{cloud,j}$ that might damage another transaction still residing in the local DB.

To illustrate the proposed algorithm for the second model, consider the following example: Suppose that Bob has just arrived at the emergency room with a medical issue.

Algorithm 3.4 Damage Assessment Algorithm for the Second Model

Input:

- Malicious transactions list MT_L, Aff_Lfmx$_j$ from another fog, or Aff_L$cloud, j$.
- The local log file.

Output:

- List of all affected transactions Aff_Lfm$_j$.
- Sub-lists of affected transactions for each FM$_x$ that has been read any damaged data from FM$_j$.

The Algorithm:

Once FM$_j$ receives any of the input list

1: Creates a new affected list Aff_Lfm$_j$ and initializes to null where j is the current fog node ID.
2: Copy all malicious / affected transactions T$_i$ that exist in FM$_j$ from the received list to Aff_Lfm$_j$ list
3: Scan the local log and
4: **for** every T$_k$ $\in$ FM$_j$ that is dependent on any T$_i$ $\in$ Aff_Lfm$_j$ **do**
5: Mark T$_K$ as affected transaction
6: Add T$_K$ to the Aff_Lfm$_j$ list
7: **end for**
8: **for** every T$_i$ $\in$ Aff_Lfm$_j$ **do**
9: **if** T$_i$ has been read by any other FM$_X$ **then**
10: Check If Aff_Lfmj$_X$ list does not exist, Then
11: Creates a new affected list Aff_Lfmj$_X$ and initialize to null where x is the id of aimed Fog node that reads the affected transaction.
12: Add T$_i$ to the Aff_Lfmj$_x$ list
13: **end if**
14: **end for**
15: Send Aff_Lfmj$_X$ to proper FM$_x$ to do further detection
16: Send Aff_Lfm$_j$ for data recovery.

All of Bob's data will thus be sent to the Fog of the Emergency Room (Fog$_{ER}$). A few hours later, however, the doctors decided to move Bob to the Intensive Care Unit (ICU) because his heart rate was not steady according to sensor S$_1$ (call this transaction T$_1$). Bob's data will then also be moved to the Fog for the Intensive Care Unit (Fog$_{ICU}$). While Bob was in the ER, the physicians performed their tests and wrote report (T$_2$) and, based on that report, the nurses gave Bob some antiarrhythmic medication (T$_3$) to stabilize his heart rate.

Thus, transaction T_2 is dependent on transaction T_1, and T_3 is dependent on T_2. Suppose, also, that medication T_3 has some side effects, such as abdominal pain (T_4), hemoptysis (T_5), or hypoglycemia (T_6). These three transactions should be taken into account.(This is an example of why affected and damaged transactions should not be deleted, but instead kept and marked as affected). Once Bob arrives in the ICU, he develops a reaction to the antiarrhythmic medication T_3. The critical care specialist, based on the data and reports T_1, T_2, and T_3 from the ER, decides to give Bob alternative antiarrhythmic medication T_{ICU1}.

Subsequently, the critical care specialist realizes that Bob has no heart issue and his heart rate is normal. In the meantime, however, the IDS has notified the Fog_{ER} that T_1 is a malicious transaction. There is then no way for Fog_{ICU} to detect T_2 and T_3 as affected transactions, since they depend on T_1 and they are done locally at Fog_{ER}. Nevertheless, Fog_{ER} will be the only fog node in the entire system that can detect T_2 and T_3. Thus, when Fog_{ER} receives the malicious transaction list including T_1, the model's first procedure will confirm whether T_1 has been flushed to the cloud. Suppose that T_1 is still in the local DB and has not been flushed yet. Fog_{ER} will then launch Algorithm 3.4, as described in the second procedure. The first line of the algorithm will create the Aff_LFM_{ER} list to identify all affected transactions in the Fog_{ER}. T_1 will be copied to this list, since it still resides in the local DB. Next, lines 3 - 7 will scan the log file, starting from T_1, and check each transaction sharing data dependency with T_1 or with any other transactions in the list. The algorithm will find T_2 in this way, mark it as an affected transaction, and add it to Aff_LFM_{ER}. When the algorithm examines the next transaction in the log for data dependency with any affected transactions from Aff_LFM_{ER}, it will now take T_2 into consideration because T_2 has just been added to the list. Thus, T_3 will be detected, marked as affected, and added to

Aff_LFM$_{ER}$.

After all transactions in the log file have been scanned and examined, lines 8 - 12 of Algorithm 3.4 will confirm whether each affected transaction has been read by another fog node (to which it was moved when the patient was moved). If so, a sub-list of affected transactions will be created for each affected fog node, and add to the list the affected transaction that has been read. In our example, transactions T_1, T_2, and T_3 have been moved to Fog$_{ICU}$, so Fog$_{ER}$ will create a new sub-list Aff_LFMER$_{ICU}$ and add T_1, T_2, and T_3 to that sub-list. Finally, Aff_LFMER$_{ICU}$ will be sent to the Fog$_{ICU}$ for further detection, and the main affected list Aff_LFM$_{ER}$ will be sent for data recovery. Once Fog$_{ICU}$ receives Aff_LFMER$_{ICU}$, it will use the list as input in the same algorithm, and T_{ICU1} will be detected as an affected transaction, since there is data dependency between T_{ICU1} and the affected transactions T_1, T_2, and T_3. The process continues until all affected transactions are detected.

3.6 Experiments and Evaluation

3.6.1 Setup and Introduction

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 quality of the proposed algorithms was evaluated by performing experiments considering different factors, including the number of transactions in each log file and the number of attacking transactions and fog nodes in the system. The communication delay between fog

nodes in the experiments was ignored since we used a local personal computer.

For the first model, we implemented different factors in order to draw a comparison between the two proposed models: the fog model using heterogeneous data, and the fog model using homogeneous data. We calculated the total time required, from initial detection of malicious transactions by the (IDS), to assess and detect all affected transactions in the system. Furthermore, we recommended the model most suitable, or proper, considering circumstance and other factors, for use in the healthcare system or a similar environment .

For the second and third models we simulated two primary systems. The first system represents distribution by private, trusted fog nodes. The second simulated system represents distribution by public, non-trusted fog nodes. Then we applied our proposed damage assessment and data recovery schemes, with minor modification, to both so that we might study the differences between them. Our goal is to show which model is more efficient, again considering circumstance and other factors, for use with each scheme.

The factors we used for all models are as follow:

1. Number of transactions in each log files:
    - 100 transactions
    - 500 transactions
    - 1000 transactions
2. Set of affected transactions in the whole system:
    - Less than 5 affected transactions.
    - 10 to 15 affected transactions.

- 30 to 35 affected transactions.

3. The number of fog nodes in each model

   - 5 fog nodes.
   - 10 fog nodes.
   - 15 fog nodes.
   - 20 fog nodes.

3.6.2 Simulation

We simulated the whole environment for each proposed model to prove the applicability and efficiency of our models and algorithms. Then we evaluated the quality of the proposed algorithms by experimenting with different factors such as number of transactions in each log file, the number of affected transactions in the whole system, and the number of fog nodes in the system. We sought answers to the following:

- How efficient are the proposed damage assessments algorithms to detect all affected transactions and data items in the systems.
- Compute the total time for each proposed damage assessment algorithm once the malicious transactions list has been received until all affected transactions have been identified. During this time the system will be unavailable so, we aim to minimize this time as much as possible.
- How efficient are the proposed data recovery algorithms to return the damaged data to consistent state.

- Compute time latency for each proposed data recovery algorithm from the affected transactions list has been received until the whole data items have been recovered. During this time only the affected transactions and data items will be unavailable to be use so, we aim to minimize this time as much as possible.

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, in some models , we compare the performance of the proposed sub-models against each other to see which model is better for different circumstances.

3.6.3 Data-set

3.6.3.1 Log Files:

Since there is no available data-set for the log files of fog node computing, 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. For example, in the first model the read operations and their values, as seen previously in the section 3, will be added to the log files for that model. We performed our experiments based on the three log file sizes described above, 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) r_2(345,669) w_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 $r_6(238,689)$ $w_7(207,612,636)$ $r_8(356,641)$ $r_9(323,683)$ $w_6(238,689,604)$ $w_7(262,679,633)$ $w_8(356,641,654)$ $w_9(323,683,635)$ $w_6(345,621,636)$ 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.

3.6.3.2 Graphs

We used the adjacency list for directed graph representation due to its speed and space efficiency technique [79]. 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

[80]. 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 informatic 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 Chapter 5. 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, as we elucidated in Chapter 5. The log files and graphs were manually rechecked and examined as ground truth to ensure the correctness and accuracy of the algorithms.

3.6.4 Evaluation First Model: Ensuring Data Integrity in Fog Computing Based Healthcare Systems

In the healthcare systems model, we introduced two sub-models for fog computing systems that manage healthcare data. The first one is an architecture using fog computing with homogeneous data; the second one is an architecture using fog computing with heterogeneous data. Then for each sub-model we proposed appropriate algorithms to determine the effect of an attack on the system and identify damaged data.

In this experiment our goal was to determine two important points. First, for each model we needed to study the behavior of the damage assessment algorithm as it detected affected data items with varying factors designed for the experiment. Second, we performed experiments to highlight the differences between the two models. Which is the better performing of the two and which factors influence that outcome?

As we mentioned in the simulation section, we generated different sets of log files. Each set represented the number of fog nodes in the system with a different number of transactions in the log file. So, in the experiment we began with a fixed number of fog nodes each time and a different number of transactions in each log file. Each time a malicious transaction was randomly inserted the results were classified into three different sets, of affected transactions. The first set is a set of less than five identified, affected transactions. Ten to 15 identified, affected transactions made up the second set. The third set of identified, affected transactions numbers twenty to twenty-five. This transaction clustering is important in making a fair and reasonable comparison of the proposed algorithms as they are impacted by other factors including the number of fog nodes and the number of transactions in each log

file. Each transaction in each set was repeated approximately twenty times. Then the total time, from insertion of the malicious transaction until all affected data items in the system were identified, was computed. The time required for each set to identify the damaged transactions was then averaged. This average was calculated for use in investigating and evaluating our approach. We compared the results, determining the factor or factors having a greater or lesser impact to the algorithms.

- **First Sub-Model "Fog Node Distribution with Homogeneous Data"**

3.6.4.1 First Experiment: The impact of log file size on different number of fog nodes in the homogeneous data model:

Figs. 3.3, 3.4, 3.5, and 3.6 explain the relationship between the number of transactions in each log file, the number of affected transactions, and the average time required by our system to detect all affected data items in the whole system. The average time was calculated by averaging the detection time each set required to complete the tasks. The results show that a log file of 100 transactions will take the system approximately 34 to 36 ms to detect all affected data items in the whole system no matter the number of affected transactions. The delay time in detecting all affected data items in log files of 500 transactions increased by approximately 21- 23 ms. The delay time for the system to identify all affected transactions in log files of 1000 transactions was only around 17 to 18 ms more than that, when compared to 500 transaction log files, and almost the double the time the system required to identify the affected transactions in log files of 100 transactions. So, the length of time the system remains unavailable is considerably impacted by the number of transactions in the log files. This applies regardless of the number of fog nodes or the number of affected transactions in

each log file.

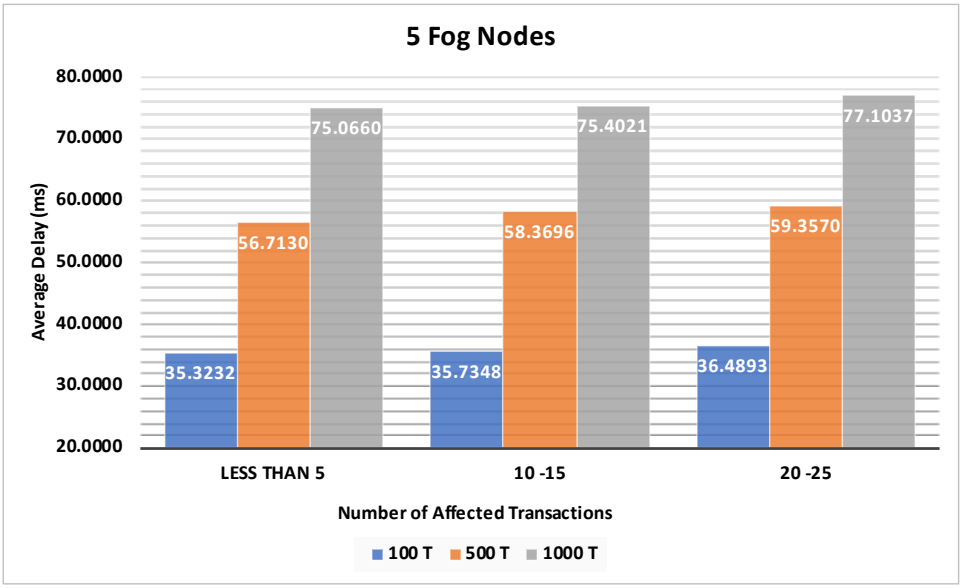

Figure 3.3: The impact of log file size on five fog nodes in the homogeneous data model.

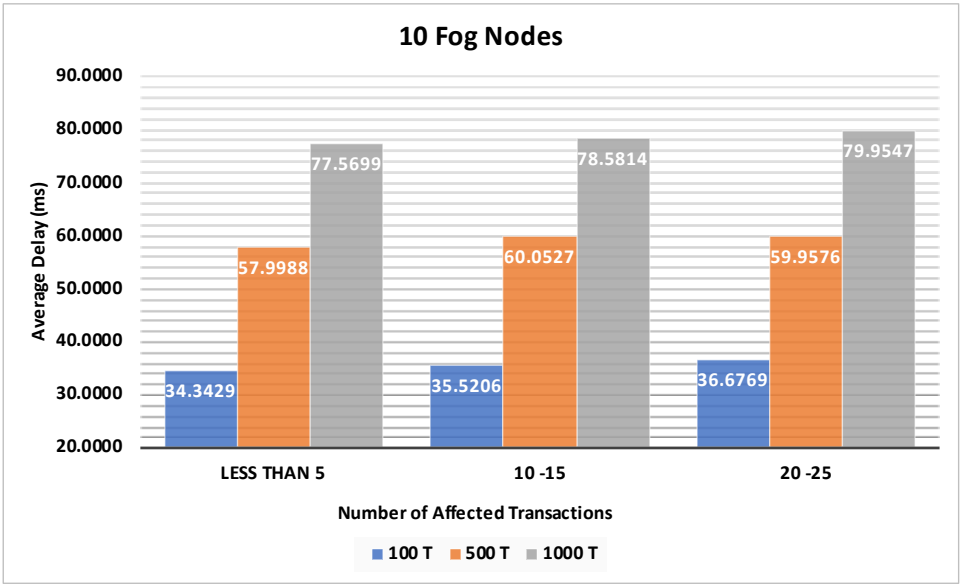

Figure 3.4: The impact of log file size on ten fog nodes in the homogeneous data model.

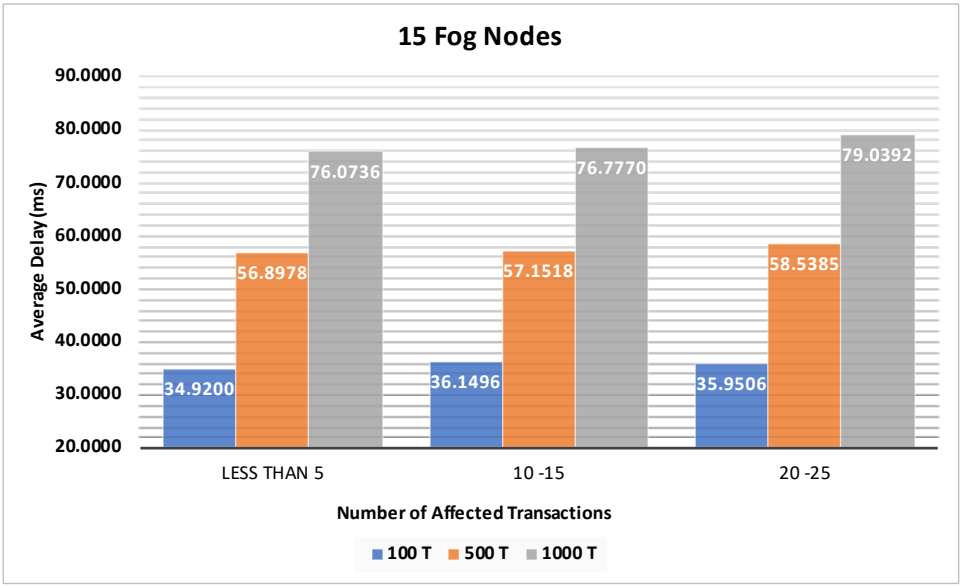

Figure 3.5: The impact of log file size on fifteen fog nodes in the homogeneous data model.

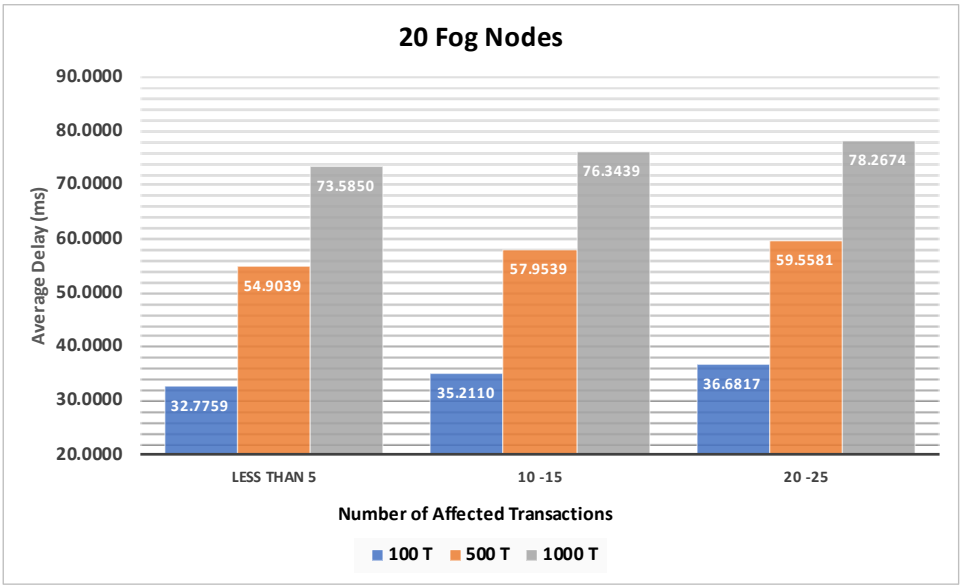

Figure 3.6: The impact of log file size on twenty fog nodes in the homogeneous data model.

3.6.4.2 Second Experiment:The impact of a variable number of fog nodes and sets of affected transactions on various size of log files in the homogeneous data model:

This experiment was designed to evaluate the performance of our detection algorithm in identifying all data items affected by the introduction of a malicious transaction. Figs. 3.7, 3.8, and 3.9 summarize the relationship between the number of fog nodes and the sets of affected transactions. Again, each log file was fixed in sets of 100, 500, and 1000 transactions. The goal of this experiment was to discover and study the impact of the different sets of affected transactions and the various number of fog nodes to our damage assessment algorithm.

Figs. 3.7, 3.8, and 3.9 show that our algorithm performs essentially the same regardless of log file size. The damage assessment algorithm detected all affected data items with a marginal change in time required to complete the task, either increasing or decreasing by less than 0.5 ms, when the number of fog nodes increased. This held true regardless of log file size. Furthermore, the number of affected transactions in each set had only slight impact on delay time: at roughly 0.5 to 1 ms.

Figs 3.7 through 3.9 further illustrate that the set of five affected transactions entailed less time than the other two sets for all variants of log file size and number of fog nodes. Additionally, the set of ten to fifteen affected transactions, for the most part, exacted less time than the set of twenty to twenty-five. The little change that occurred here was due to the algorithm computation.

In brief, these experiments have shown that the earlier IDS identifies malicious trans-

actions, the more efficient and faster damage assessment can be made and that the size of the log file has the most impact on delay time.

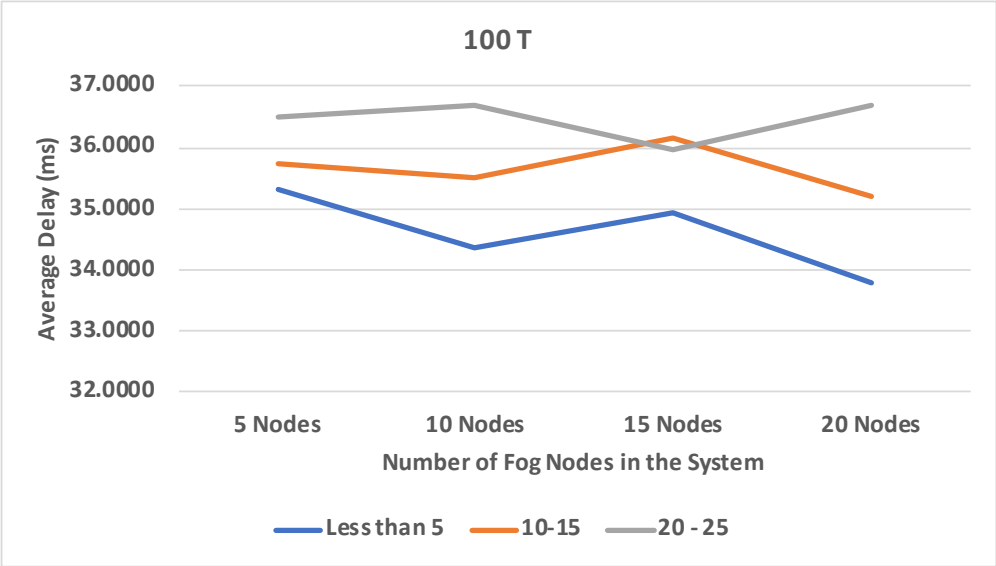

Figure 3.7: The impact of a variable number of fog nodes and sets of affected transactions on log files of 100 T in the homogeneous data model.

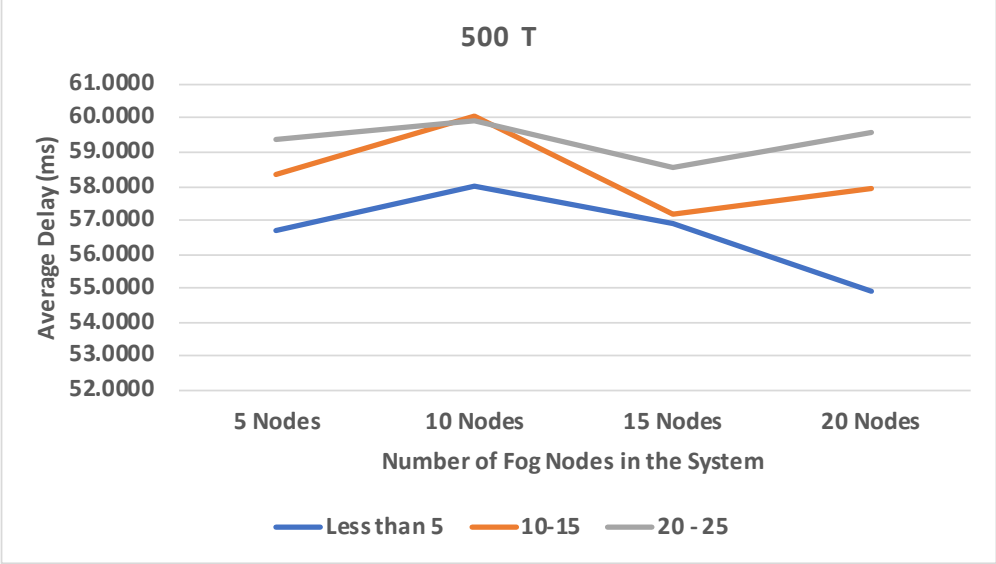

Figure 3.8: The impact of a variable number of fog nodes and sets of affected transactions on log files of 500 T in the homogeneous data model.

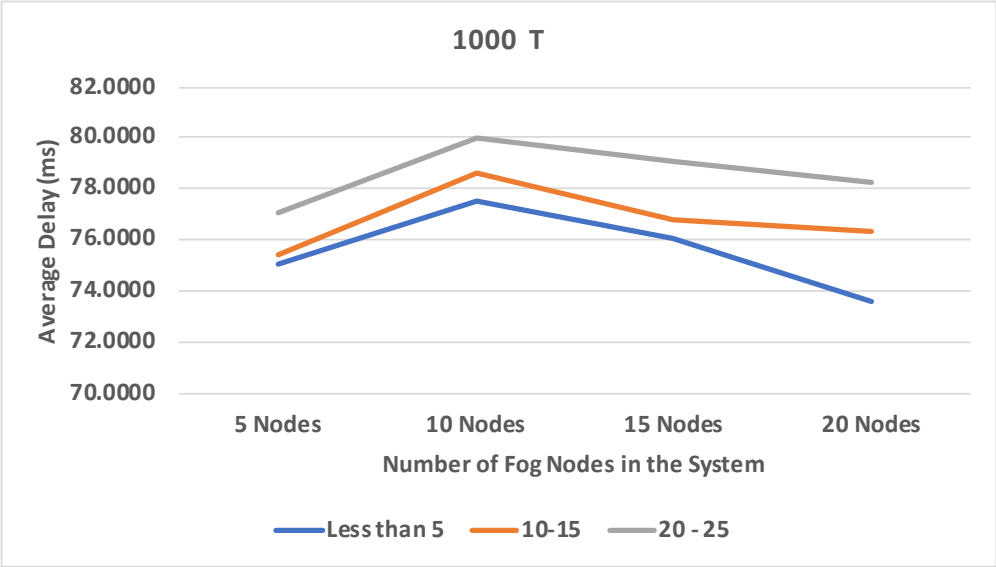

Figure 3.9: The impact of a variable number of fog nodes and sets of affected transactions on log files of 1000 T in the homogeneous data model.

- **Second Sub-Model: "Fog Nodes Distribution with Heterogeneous Data"**

3.6.4.3 Third Experiment: The impact of log file size on different number of fog nodes in the heterogeneous data model:

This experiment uses the same variables as the experiment described in section 3.6.4.1. We computed the average time the algorithm exacts to identify all affected data items in the whole system. The number of fog nodes was fixed to five, then to ten, fifteen, and twenty, respectively. The numbers of transactions on each log file varied from 100 to 500, then to 1000. The results were classified in different sets as indicated in Figs. 3.10, 3.11, 3.12, and 3.13. The results show that a 100 transaction log file, results in a delay time around 36 to 42 ms for detection of all affected data items in the whole system for less than five affected transactions. And the delay time to detect the same set in a 500 transaction

log file increased to 51 to 79 ms. A 1000 transaction log file with fewer than five affected transactions, rendered a delay time of 100 to 119 ms in identification of all affected data items. This increase in delay time as it relates to the number of transactions in the log file remains consistent regardless of the number of fog nodes or affected transactions in each log file. As the number of transactions in the log files has a major effect on delay time, system availability becomes further compromised as the transaction count per log file increases.

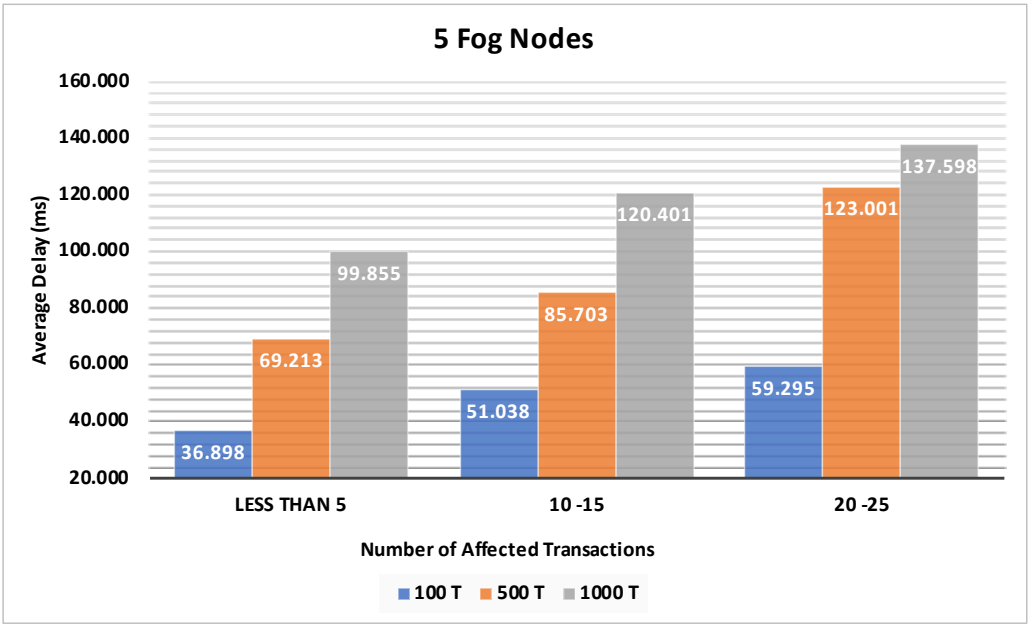

Figure 3.10: The impact of log file size on five fog nodes in the Heterogeneous data model.

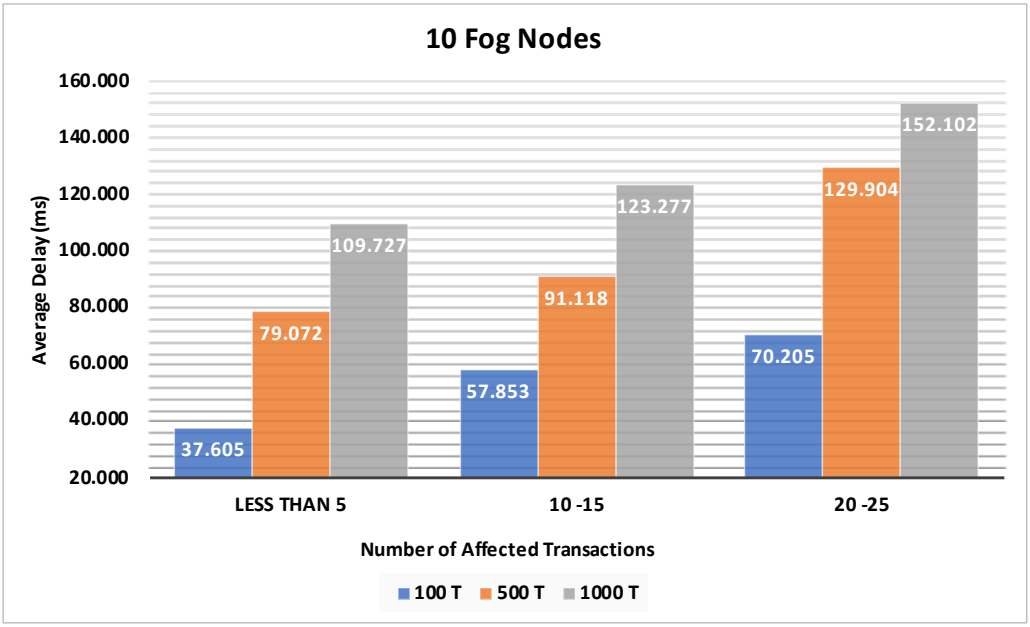

Figure 3.11: The impact of log file size on ten fog nodes in the Heterogeneous data model.

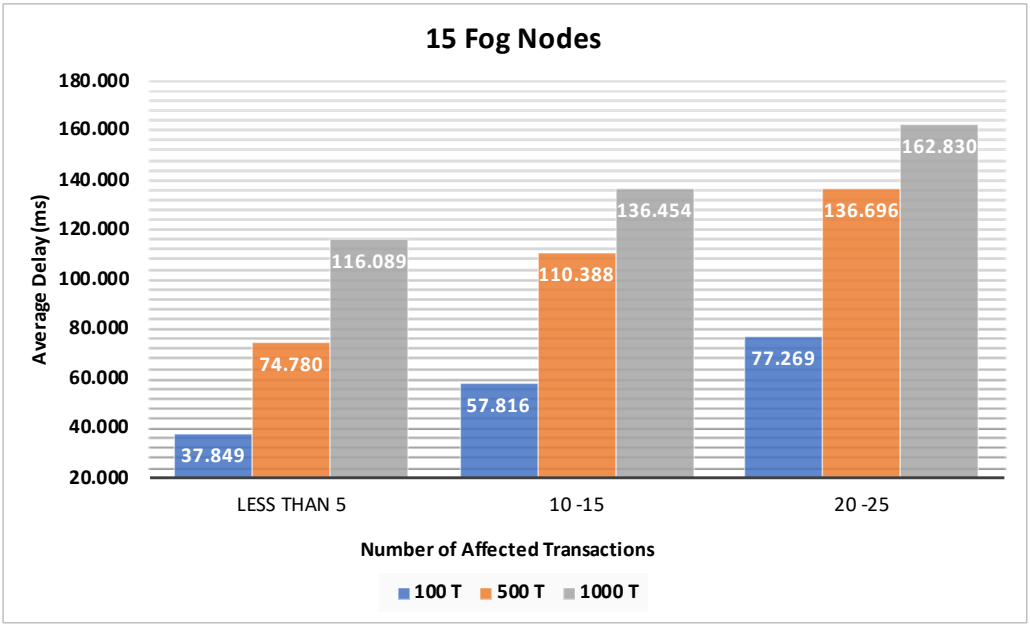

Figure 3.12: The impact of log file size on fifteen fog nodes in the Heterogeneous data model.

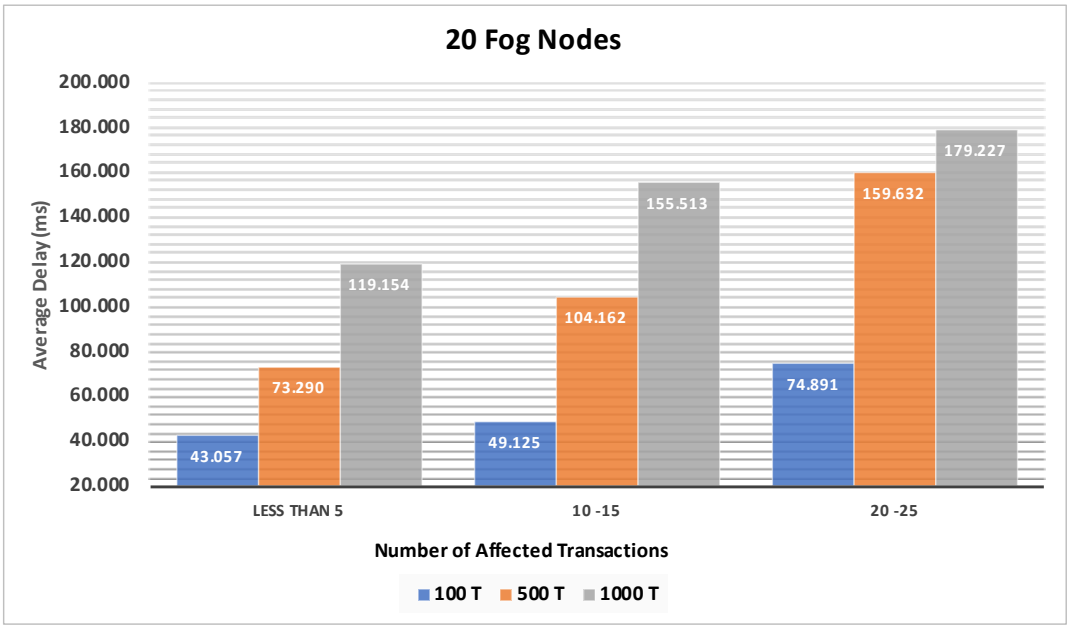

Figure 3.13: The impact of log file size on twenty fog nodes in the Heterogeneous data model.

3.6.4.4 Fourth Experiment: The impact of different sets of affected transactions on various number of fog nodes in the heterogeneous data model:

The second experiment was implemented to observe the impact of the different sets of affected transactions in the heterogeneous data model. Figs. 3.14, 3.15, 3.16, and 3.17 represent the results. The findings observed in this experiment are different than that of the experiments in section 3.6.4.2 for the homogeneous data model. The results here show that a log file of 100 transactions produces, a delay time in ms that jumped from the thirties for detecting the set of less than five affected transactions to the fifties for ten to fifteen affected transactions, and into the seventies for twenty to twenty-five affected transactions. The delay time to detect the affected data items in a log file of 500 varied from the small set of affected transactions to the larger set, ranging from ms in the 70's for the set of less than

five affected transactions to ms in the 150's for the set of 20 to 25. The 1000 transaction log files registered a delay time around the 100's to 170's ms to identify all affected data items . So, unlike the homogeneous data model, in this model the number of affected transactions in the system has significant effect on the total time the system remains unavailable.

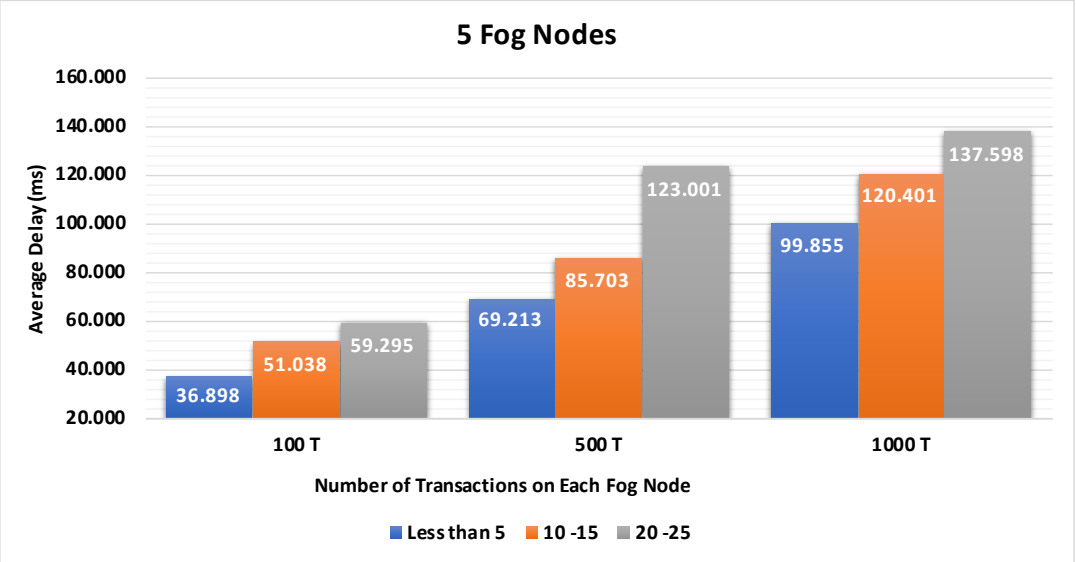

Figure 3.14: The impact of different sets of affected transactions on five fog nodes in the heterogeneous data model.

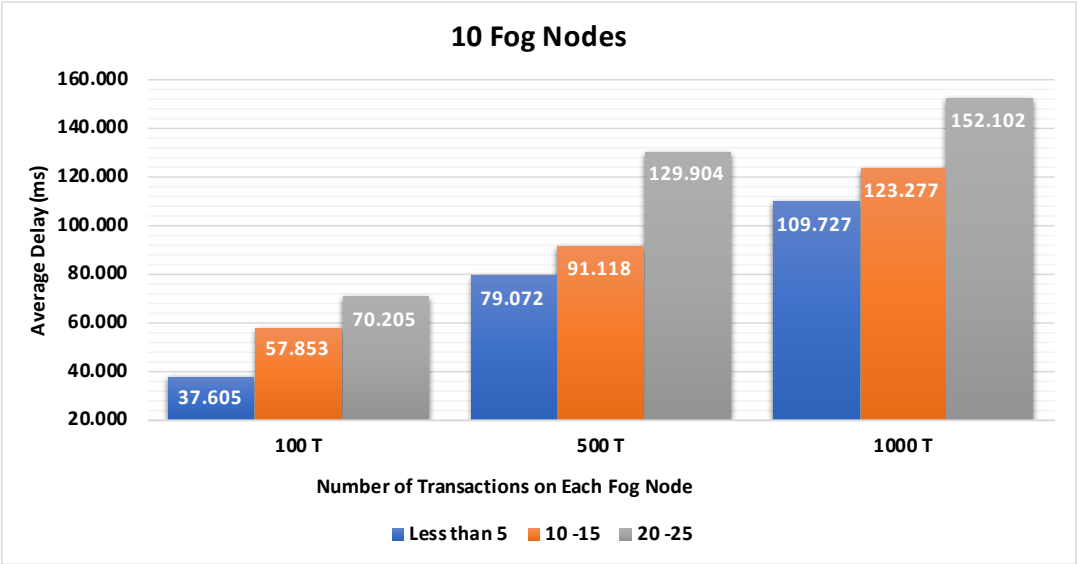

Figure 3.15: The impact of different sets of affected transactions on ten fog nodes in the heterogeneous data model

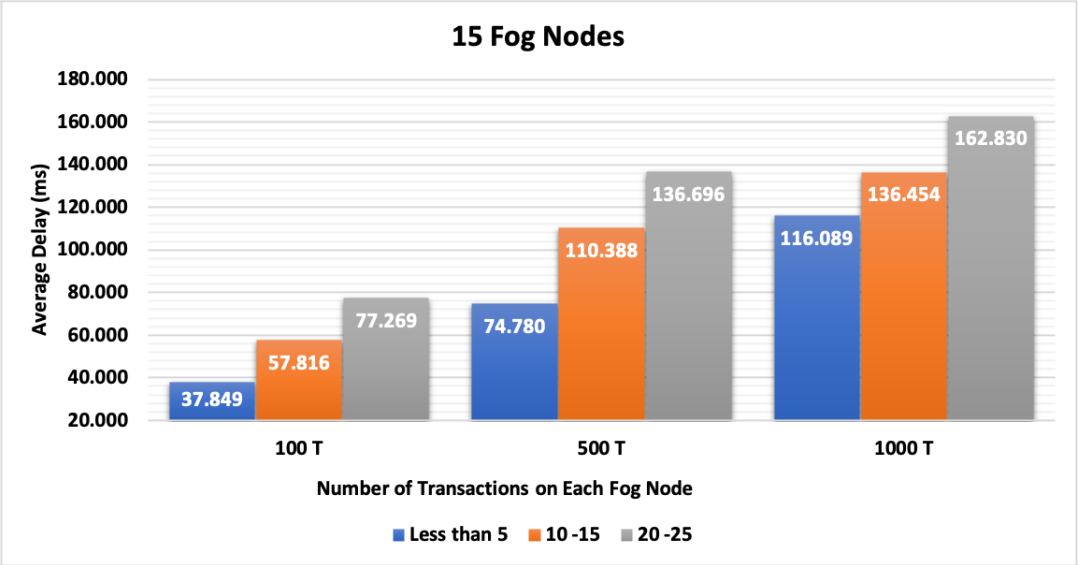

Figure 3.16: The impact of different sets of affected transactions on fifteen fog nodes in the heterogeneous data model.

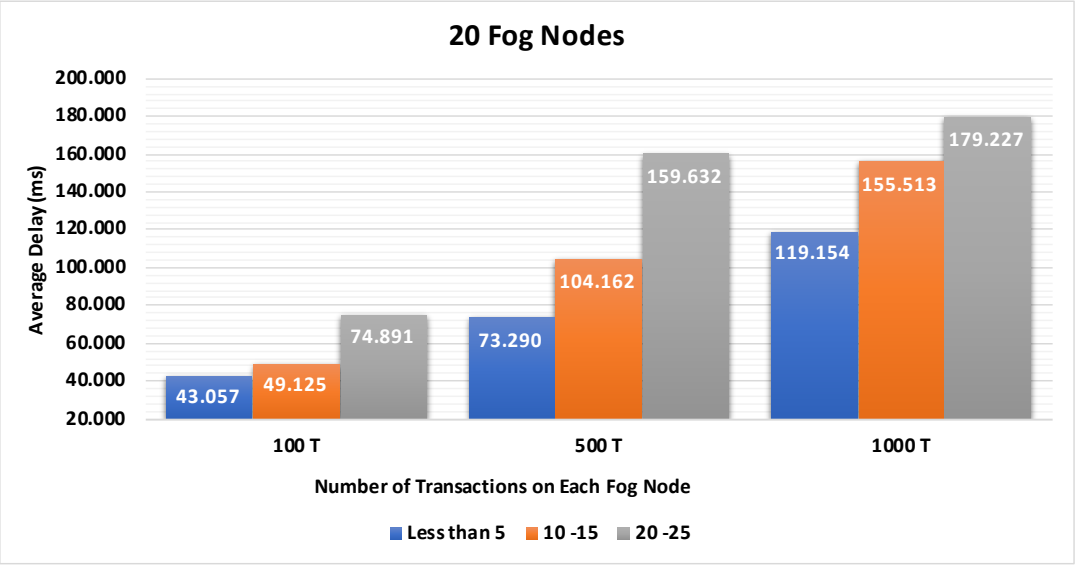

Figure 3.17: The impact of different sets of affected transactions on twenty fog nodes in the heterogeneous data model.

3.6.4.5 Fifth Experiment: The impact of a variable number of fog nodes and sets of affected transactions on various size of log files in the heterogeneous data model:

Investigation was also conducted into the relationship between the number of fog nodes and the different sets of affected transactions. Figs. 3.18, 3.19, and 3.20 describe the results. The number of transactions on each log file here was fixed to 100, then 500, and finally to 1000. The objective was to study the impact of the different sets of affected transactions with a variable number of fog nodes while keeping a fixed number of transactions in the fog file. As we see in these figures, the performance of the proposed algorithm generated roughly the same outcome with all different log file sizes. We found that the time needed for our damage assessment algorithm to detect all affected data items increased by 1 - 10 ms, when the number of fog nodes increased. This remained consistent regardless of log file size.

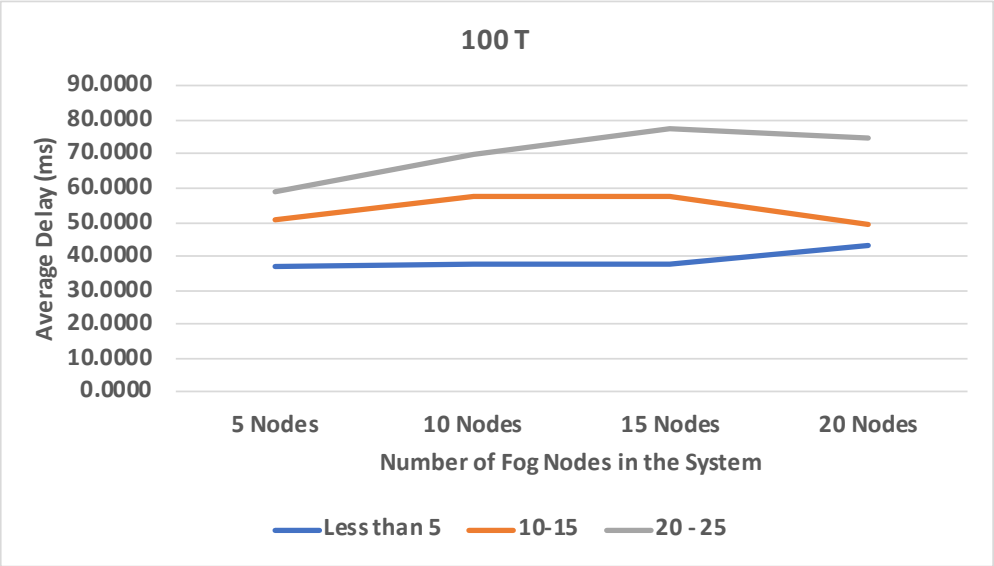

Figure 3.18: The impact of a variable number of fog nodes and sets of affected transactions on log files of 100 T in the heterogeneous data model.

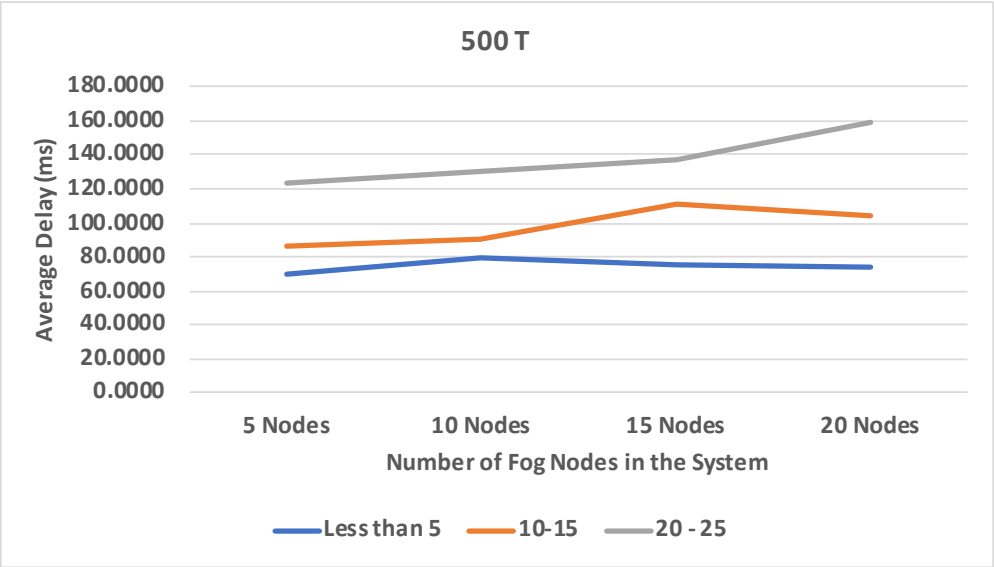

Figure 3.19: The impact of a variable number of fog nodes and sets of affected transactions on log files of 500 T in the heterogeneous data model.

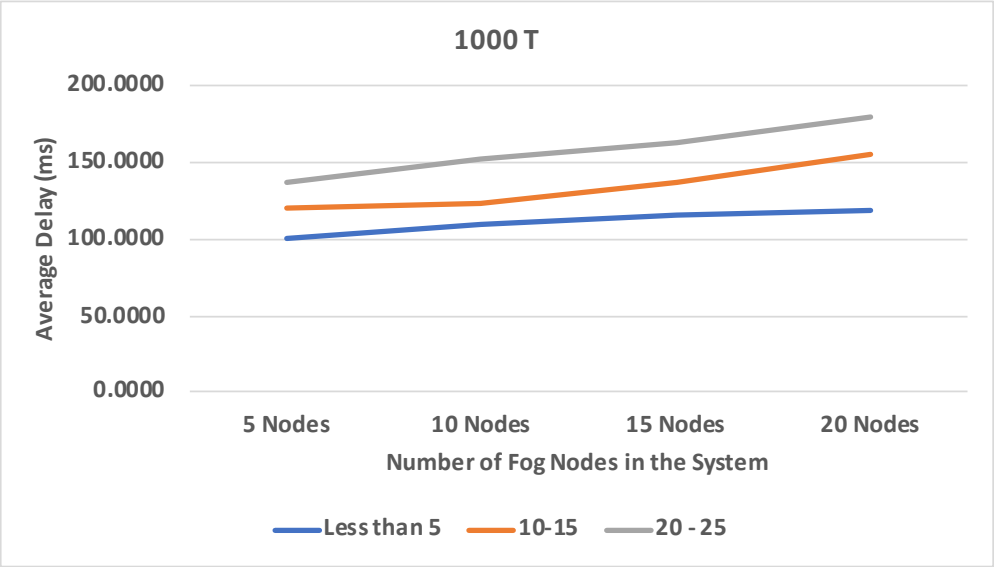

Figure 3.20: The impact of a variable number of fog nodes and sets of affected transactions on log files of 1000 T in the heterogeneous data model.

3.6.4.6 Sixth Experiment: Fog Nodes Distribution with Homogeneous Data Model versus Heterogeneous Data Model:

These experiments were executed to demonstrate the difference between the two proposed models and to discover, in consideration of all variables, which of the two is the better performer. Figs. 3.21, 3.22, and 3.23 illustrate an overall comparison of the average total time latency in milliseconds between these models . The result, supported by these figures, indicates that the first model, the homogeneous model, performs better with all different sets of affected transactions.

However, with the smaller set of less than five affected transactions and smaller log file of 100 transactions, the difference was slight; the first model exceeded the second by only 1 - 3 ms (Fig. 3.21). With the same smaller set of less than five affected transactions the difference between the two models was a more significant 10 to 20 ms when the number of

transactions increased from 100 to 500. Increasing the size of log files to 1000 transactions again rendered a significant difference of 20 to 30 ms. Figs. 3.22 and 3.23 illustrate the clearly meaningful difference between the two models with the introduction of the two larger sets of affected transactions. Briefly, the first model required less time than the second model, by half in log files of 500 transactions, and by a third in log files of 1000 transactions.

The explanation for that lies in the number of fog nodes that will be affected by the malicious transaction. With the small set of less than five affected transactions, the damage will usually only affect an average of up to three fog nodes. In the worst case, the architecture of the first model will prevent the damage of one malicious transaction from spreading to more than two fog nodes .

But in the second model the larger set of affected transactions could impact a greater number of fog nodes. So each log file in each affected fog node must be scanned in the second model, while the first model need only scan two fog nodes at most, the initial affected Fog Monitor node and the FDR fog node if it read the damaged data items. This scanning requirement is also the reason that causes the variations in delay time in the second model. As shown in Fig. 3.23 there are noticeable differences in average delay time from one group of fog nodes to another in the second model with 1000 transactions. For example, if a malicious transaction affected fifteen transactions in the system, then all those affected transactions could be in one fog node or in five fog nodes or more.

In closing, the model with homogeneous data stabilizes the results and speeds the detection process, minimizing the system's unavailability. Although, the model with heterogeneous data is needed as well because it is more appropriately applicable to most situations.

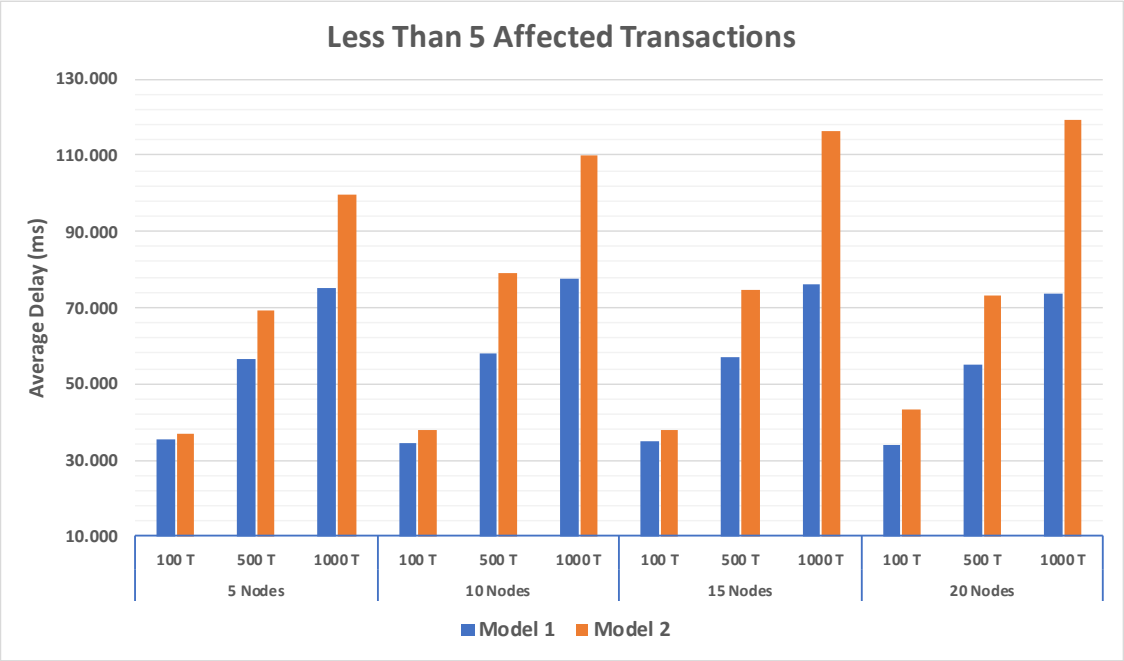

Figure 3.21: Comparison between homogeneous data model and heterogeneous data model on a set of less than five affected transactions

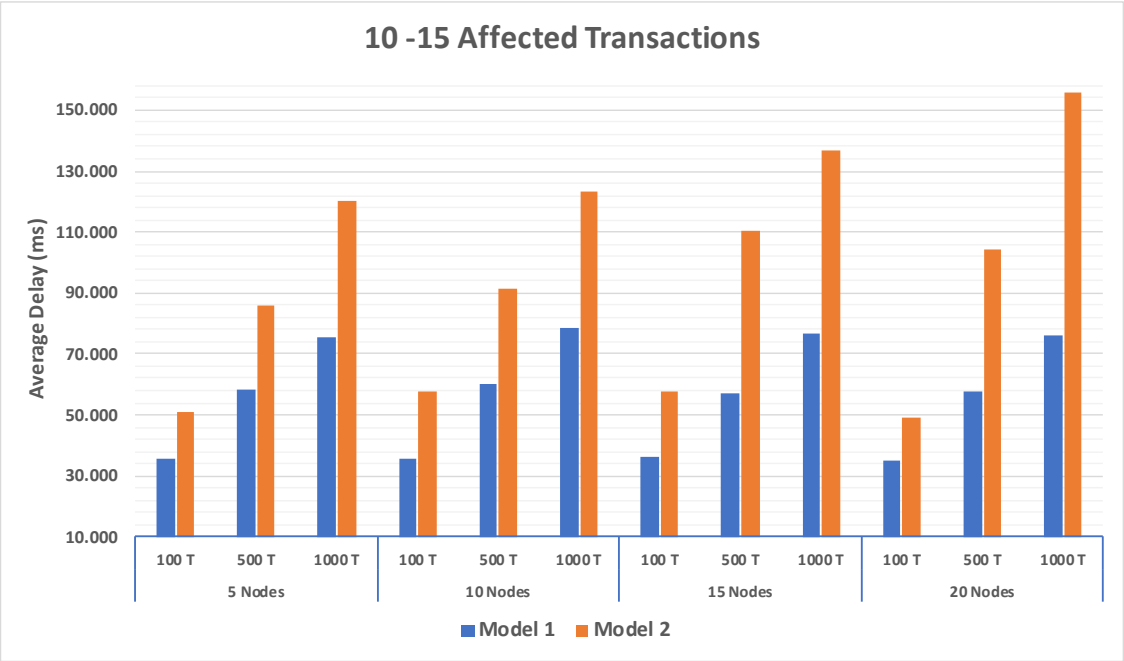

Figure 3.22: Comparison between homogeneous data model and heterogeneous data model on set of 10 to 15 affected Transactions

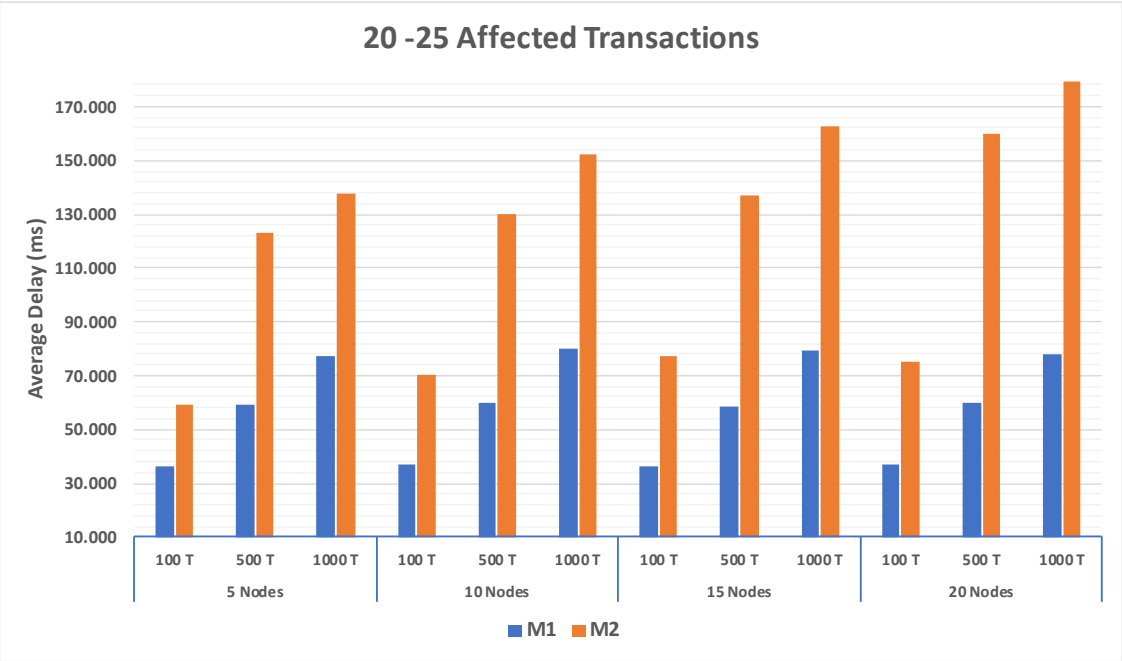

Figure 3.23: Comparison between homogeneous data model and heterogeneous data model on a set of 20 -25 affected Transactions

4 Ensuring Data Integrity in Fog Computing Based Critical Infrastructure Systems Model

Some of the following paragraphs, figures, and algorithms that will be introduced in this chapter have been already published in our work [48] as shown in the publications and reprint permissions chapter 7.

4.1 Introduction

These days the Internet of Things (IoT) is prevalent and trending as evidenced by the significant increase in wearable technology, smart homes and buildings, connected vehicles, and smart grids. The estimated number of connected IoT and smart devices by 2025 is expected to become 75 billion worldwide [1]. This will produce an enormous amount of data that are predicted to total more than 79 zettabytes [2]. Due to the limitation and restriction of bandwidth, as well as the rapid growth in the amount of data produced, the current architecture of internet systems will be inadequate to manage and move that volume of data to the cloud. In many scenarios, this will be impossible to do, especially with the increasing number of IoT devices in use. Furthermore, some sensitive and real-time applications play integral roles in our lives, such as connected car technologies, video conference applications, and health monitoring devices, all of which require low-latency and location awareness to provide satisfying and high-quality services [4].

In addition, IoT devices such as smart meters in modern smart cities will not only produce a massive amount of data but the heterogeneity of that data will need to be processed

in real-time [23]. This data will not be valuable enough without exploiting the maximum benefits from other technologies. It is unmanageable and nearly impossible for the cloud to handle by itself so many kinds of tasks, such as processing the aggregate data, analyzing them, and storing them [24]. For that reason, fog computing was presented by Cisco [6]. Fog computing is a decentralized computing infrastructure that allows end users' data to be computed, stored, and promoted on other applications and network services that are located between the data source and the cloud computing data centers.

Fog computing technology is trending these days because it has several characteristics that can improve the efficiency of transporting data to the cloud. This is very significant in the age of cloud processing because it complements cloud computing and allows analytics to perform resource, intensive, and extended term analytics [10]. One of the essential characteristics of fog computing is its location at the edge of networks, providing end users with high-quality services and low latency. Many current devices and applications, such as smart meters, networked games, and video streaming, require location awareness and low latency to provide a higher quality of service and performance. Alternative fundamental characteristics are the widely dispersed and significant numbers of fog nodes that will be geographically available. This design supports mobility in many applications, including service in moving vehicles. The utilization of fog computing will also contribute to the benefits of conserving bandwidth and enhancing the privacy and security of sensitive data. Most of the data will be processed locally at the fog node, meaning that the amount of data needing to be sent to the cloud for processing will be diminished, thereby minimizing bandwidth consumption and maximizing the privacy of transmitting sensitive data [8, 9].

In the potential for smart cities, fog computing will help the IoT and smart meter

devices process the data and make quick decisions to take the right action within a critical time-frame and to aggregate only the indispensable data for the cloud. Services and utility companies, such as water and electric companies, can exploit fog computing technology to manage and analyze the volume of consumers' data. There are many existing studies on how to expand the efficiency of fog computing in smart cities and to solve technical issues related to the large volume of data that needs fusion and integration to the cloud [22]. Moreover, while security and privacy issues have been addressed by many researchers, other aspects need more attention, such as a case in which a protection system fails during a cyberattack and costumers' data need to be recovered. This study aims to detect all transactions that are affected by any malicious transaction, recover the correct value of data, and ensure the integrity of consumer data in a fog computing environment in smart cities.

4.2 The Model

In this section, a unique architecture for using a distributed fog node system in smart cities to manage the consumer data of utility services will be proposed. Then, cooperative algorithms will be proffered for identifying, assessing, recovering, and restoring all the damaged and affected data created by an attack. The goal is the restoration of a reliable database. In the proposed model, it is assumed that the Intrusion Detection System (IDS) is responsible for detecting malicious transactions in the system and providing a list of those transactions to the damage assessment algorithms. Each fog node in the proposed architecture must have its own log file and use a strict serializable history. All operations in the log file need to be in the same order in the history. The log files cannot be user modified at any time. Since the log files will contain a record of every modification to the value of

any data item that is updated by write operations, all read operations are also required to be stored in the log files to identify the data dependencies between the operations and the transactions and among the detections of the victim fog nodes in the systems.

4.3 Model Nations

Table 5.1 shows a list of the notations and descriptions to be used in our proposed approach.

Table 4.1: Notation used in our proposed approach description

Notation	**Description**
pub_fog	The public fog nodes that are accessed by customers and utilities providers.
usc_fog	The private fog node for each utility service company.
MT_L	The list of detected malicious transactions done by IDS.
DA_Table	The damage audit table, which is a data structure that will be created by the damage assessment algorithms to collect any data about transactions that are needed to do the data recovery, such as the valid and invalid read data items, data written, and the accessed fogs.
DI_L	The damage item list that will contain all damaged data items that are identified by our proposed damage assessment algorithms.
DIT_ Fog_x	The Fog_x damage item table, where x is the ID of the secondary affected fog node, which reads some of the damage data items from another fog node.
VIT_ Fog_x	The valid data items table that will be created by algorithms 3 or 4 to add to it all recovered data items for the secondary affected fog node Fog_x. It will be sent to Fog_x to use it as input on algorithm 4.
$w_i(A, v1, v2)$	The write operation of the transaction Ti; v1 is the before image, which represents the old value of the data item A before the update. And v2 is the after image, which is the new value of data item A after the update.
$r_i(A, v)$	The read operation of transaction T_i where A is the data item and v is the current value of A.
c_i	The transaction T_i has been committed which means it is successfully completed and it will be recorded to the database.
a_i	The transaction T_i has been aborted and it will not affect the database.

4.4 The Proposed Architecture

In the proposed model, each smart city will have several public fog nodes (pub_fog), which will be efficiently distributed to guarantee the quality of service at each point and in all corners of the city. Private fog nodes will be included, with at least one private fog node for utility service companies (usc_fog), such as water, electricity, and gas utilities. The usc_fog nodes should be effectively located in the center of the whole distributed system to ensure a reliable connection to all pub fog nodes and provide different routes should one of the pub_fog nodes disconnect for any reason. Consumers will be able to send queries to pub_fog nodes only. Data may be retrieved from the local database if available there. Otherwise, the queries will be forwarded to the appropriate usc_fog node. Consumers are not allowed to directly connect the usc_fog nodes for security reasons. All queries related to those nodes will come through the pub_fog nodes. Customer utility usage data will be collected from smart homes and buildings using IoT devices and smart meters. Usage data will be sent to the nearest efficient pub_fog nodes based on several factors, such as location and load balance.

It is assumed that each pub_fog node in the system will have the ability to perform some essential data operations, such as calculating customer average usage over a specific time frame or aggregating the totals of selected data values. Those operations are fundamental to optimization of the network bandwidth since the data sent over the network will be diminished by aggregating the necessary data.

Additionally, as most customer data will be processed locally, at the edge of the network, it will enhance privacy and security by reducing sensitive data transmittal. Each

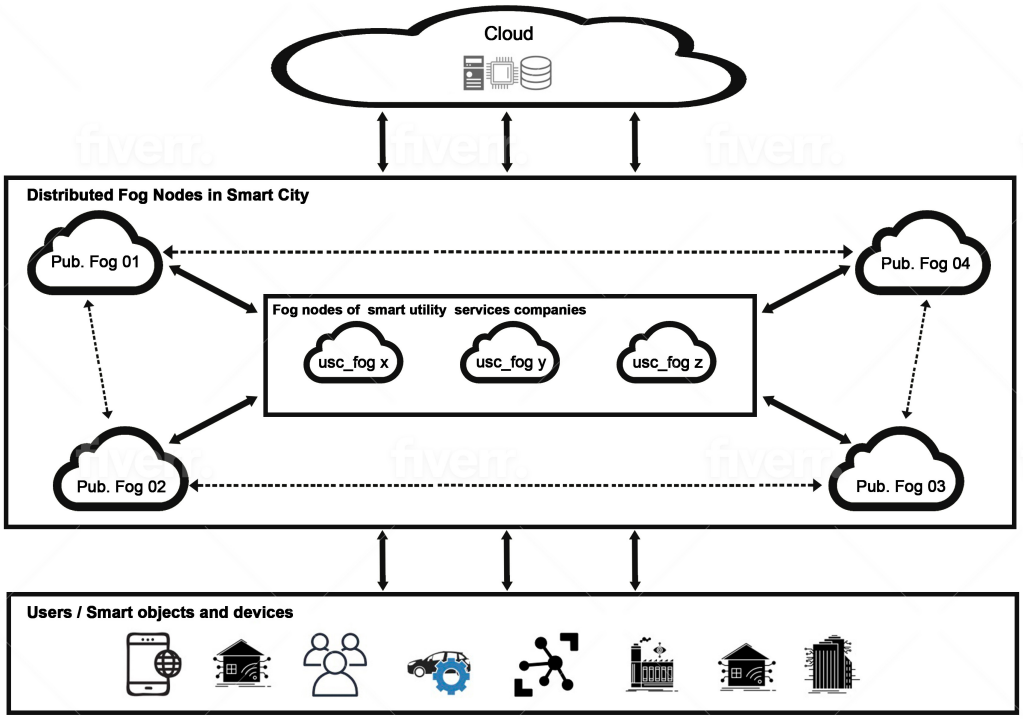

Figure 4.1: The proposed architecture.

utility usc_fog node receiving the data will also perform some essential computations, such as calculating the daily bill and average daily customer usage. These computations by the utilities are important in improving the quality of services in each city as the need for expansion of services in peak seasons may become evident and shortages avoided. Utilities may use data to plan fuel purchases or raise consumer awareness regarding consumption and conservation. We assume that the city and all fog nodes in the system are in the same time zone and use the same official local time.

4.5 The Proposed Damage Assessment Algorithms

4.5.1 Algorithm 1: The Main Damage Assessment Algorithm

The IDS is responsible for identifying the attacking transactions and sending a list of them to the victim fog node. Whenever one or more malicious transactions are found on any fog node in the system, the IDS will detect them and send them as a list (MT_L) to that fog node to be used as input in the proposed schemes. Once the fog node receives the list, it will launch Algorithm 1, which is the main damage assessment algorithm.

As soon as Algorithm 1 is launched, it will create the damage audit table (DA_Table) and damage data item list (DI_L). Both will be initialized to null. Then, the algorithm will scan the local log file of the victim fog node, Fog_1, beginning from the first attacking transaction of (MT_L) list, T_i. T_i will be added as a new record into DA_Table since it is the first attacking transaction. If the attacking transaction updates at least one data item, then this data item will be damaged, and any other updating transactions that read this data will be affected as well. It important to collect and store all data items that have been updated and damaged by the attacking transactions. Then, all transactions that have read those damaged data items can be identified, and data dependency can be declared between the transactions and the fog nodes in the entire system.

One of the main functionalities of this algorithm will be the collection of data before damage occurs, and store those images, which represent the pre-attack value of the data item, in the written data column on the DA_Table. These images will be used later in the recovery algorithm. Simultaneously, those damaged data items will be added to the damaged item list to determine data dependency.

Also, the algorithm will examine every transaction in the log file following, the first attack, to determine whether any other transaction is an attacking transaction, or a data access transaction from another fog node, or an updating write transaction. In the case where the transaction is an attacking transaction, the algorithm will perform as in the first attacking transaction. However, if the transaction is an access transaction from another fog node (Fog$x$), the algorithm will check every data item that has been read by Fog$x$. A new damage item table for Fog$x$ (DIT_Fog$x$) will be created and all damaged data items that have been read by Fog$x$ as well as the transaction identification and timestamp of each data item, will be added to the DIT_Fog$x$.

Since a fog node may access the same data multiple times, it is essential to know the transaction ID and time of access (timestamp); this will make it easy to find on its log file and confirm that the damaged data items were not corrected later on in the fog_x by valid updating. In the meantime, the DA_Table will be updated indicating that Fog_x has read the damaged data item, so when the recovery algorithm has successfully corrected the value of the damaged data item, it will send the correct value to Fog_x to use as input for its own recovery algorithm.

If the transaction is an updating transaction (T_w), and not an attacking transaction belonging to the malicious transaction list, then it must be added to the DA_Table and examined to accomplish two goals. The first goal is to determine data dependency. All read operations must be checked to confirm whether T_w has read any of the damaged data items that already exist to the damaged item list. If so, those damaged data items will be added to the invalid read column of T_w, and undamaged data items will be added to the valid read column. Then, all the write operations will be checked to determine whether any have read

Algorithm 4.1 The Main Damage Assessment Algorithm

```
Create a new DA_table and initialize to null
Create a new DI_L and initialize to null
for every T_i the local log starting from the first attacking transactions of MT_L do
  if T_i is attacking transaction then
    add it as a new record into DA_Table every w_i (A, v1, v2)
    add (A, v1) pair to data written column
    add A to the DI_L if it is not there

  else if T_i is transaction from another fog node x then every data item A read by T_i
    if A ∈ DI_L then
      if DIT_ Fog_x does not exist then
        Create a new DIT_ Fog_x where x is the ID of aimed fog node that reads the affected transaction
        Mark T_i as affected transaction
        Add T_i and A into DIT_ Fog_x
        Update the last column of DA_Table
      end if
    end if

  else if T_i is updating transaction then
    add it as a new record into DA_Table every r_i (A, v)
    if A ∈ DI_L then
      add A to invalid read column
      add (A, v) to valid read column
    end if

    for every w_i (A, v1, v2) do
      if invalid read column of T_i ≠ ∅ then
        add (A, v2) to data written column
        add A to the DI_L if it is not there  check If (A ∈ DI_L)
        add (A, v2) to data written column
        delete A from DI_L
      end if
    end for
    if c_i is found & (both invalid read and data written columns of T_i) = ∅ then
      delete the record of T_i from DA_Table
    else if a_i is found then
      delete the record T_i from DA_Table
    end if
  end if
end for
Send DIT_ Fog_x to Fog_x to do further detection
Send DA_Table & DI_L for data recovery (algorithm 3)
```

damaged data items. If so, that means the damage has spread and the written data item is also corrupted. Therefore, it will be added to the damaged item list, if it is not already there. However, if the transaction T_w updates any data item, (A), without reading any items from the damaged item list, then the data item (A) will be further checked evaluate its inclusion in the damaged item list. If (A) was updated without reading a corrupted transaction, that means it is a valid write and the data item (A) has been refreshed, so (A) must be removed from the damaged item list as in steps(28-30). The new value will be added to the data written column, accomplishing the second goal of adding the non-attacking updating transaction to the DA_Table. Steps (31-34) will remove the unnecessary records from the damaged audit table in the case where the T_w is aborted or both invalid read and data written columns of T_w are empty. Finally, the damage item table DIT_fog_x will be sent to fog_x to do further detection, while the damage audit table and damage item list will be sent to Algorithm 3, which is the main data recovery algorithm.

4.5.2 Algorithm 2: Secondary Fog Node Damage Assessment Algorithm

This algorithm is similar to the previous one, which is Algorithm 1, but there are some differences. The main difference in the input of this algorithm is the damage item table (DIT_Fog_x), which is one of the outputs of Algorithm 1. Let us say that fog_1 is the main victim fog node in the system, and it was attacked and maliciously updated in the transaction T_i, which wrote the data item Z. Later on, the transaction T_j indicated that fog_x has accessed fog_1 and read Z. Then, fog_x updated some other data items, such as N and M, on its database based on data item Z. Here we call fog_x the secondary affected fog node. Once the secondary affected fog node in our example (fog_x) receives the damage item

table (DIT_fog$_x$) that contains Z as a damaged data item, its transaction ID (T$_j$), and its timestamp say 9:00:00 AM, it will create a new damage audit table and initialize it to null. Note that this algorithm will use the received damage item table (DIT_fog$_x$) to store and track the damage data items instead of creating a new damage item list. The list data structure will not be supportive here since the exact time and transaction ID of the damage data items must be known.

The algorithm will scan the log file and start from the first affected transaction from the received table. Therefore, whenever an affected transaction that belongs to DIT_fog$_x$ is found, the steps (3-12) will insert it as new record to the damage audit table and check each read operations if its belong to DIT_fog$_x$ then add it to the invalid read column; otherwise, it will be added to the valid read column. Moreover, for the write operations, the updated data items along with its new values will be added to the data written column as well as they will be added to the DIT_fog$_x$ table if they are not there. In our example, data items N and M will be added to the DIT_fog$_x$ and the data written column in DA_Table. The rest of the algorithm is the same as Algorithm 1 except when any data damage is found, the time of its occurrence must be taken into account. Therefore, if fog$_x$ reads the data items M and N at any time before 9:00:00 AM and then updates some other data item value, those updating transactions will not be affected. In a like manner, the damage item table, if there is one, will be sent to fog$_y$ while the damage audit table and damage item list will be sent to Algorithm 4, which is data recovery algorithm for the secondary fog node. The process continues until all the affected transactions in the entire system are detected.

Algorithm 4.2 Secondary Fog Node Damage Assessment

Once Fog_x receives the DIT_ Fog_x
Create a new DA_table and initialize to null
for every T_i in the local log starting from the first affected transaction on DIT_ Fog_x **do**
  if $T_i \in$ DIT_ Fog_x && mark as affected **then**
    add a record for T_i into DA_Table
    for every r_i (A, v) **do**
      if $A \in$ DIT_ Fog_x **then**
        add A to invalid read column
        add (A, v) to valid read column
      end if
    end for
    for every w_i $(A, v1, v2)$ **do**
      add $(A, v2)$ pair to data written column
      add A into DIT_ Fog_x if it is not there
    end for
  else if T_i is transaction from another fog node y **then**
    for every data item A read by T_i **do**
      if $A \in$ DIT_ Fog_x & T_i Timestamp $\geq$ Timestamp of damaged data item transaction **then**
        if DIT_ Fog_y does not exist **then**
          Create a new DIT_ Fog_y where y is the ID of aimed fog node that reads the affected transaction
          Mark T_i as affected transaction
          Add T_i and A into DIT_ Fog_y
          Update the last column of DA_Table
        end if
      end if
    end for
  else if T_i is updating transaction **then**
    add it as a new record into DA_Table
    for every r_i (A, v) **do**
      if $A \in$ DIT_ Fog_x & T_i Timestamp $\geq$ Timestamp of damaged data item transaction **then**
        add A to invalid read column
        add (A, v) to valid read column
      end if
    end for
    for every w_i $(A, v1, v2)$ **do**
      if invalid read column of $T_i \neq \varnothing$ **then**
        add $(A, v2)$ to data written column

add a record of T_i into DIT_ Fog_x with data item A if it is not there check If ($A \in$ DIT_ Fog_x)
add $(A, v2)$ to data written column
delete A from DIT_ Fog_x
end if
end for
if c_i is found & (both invalid read and data written columns of T_i) $= \varnothing$ **then**
delete the record of T_i from DA_Table
else if a_i is found **then**
delete the record T_i from DA_Table
end if
end if
end for
Send DIT_ Fog_y to Fog_y to do further detection
Send DA_Table & DIT_ Fog_x for recovery (algorithm 4)

4.6 The Proposed Data Recovery Algorithms

4.6.1 Algorithm 3: The Main Data Recovery Algorithm

Immediately after Algorithm 1 has accomplished its task and sent the damage audit table (DA_Table) and damage item list (DI_L) to the main data recovery algorithm, which is Algorithm 3, all the data items in the main victim fog node will be available to be used except the damaged data items. Therefore, the availability of the system will be increased. Once Algorithm 3 receives the damage audit table, it will scan only the records of the DA_Table that reads invalid data items. Whenever a record of the data items in the invalid read column is found, the algorithm will perform three important steps. In the first step, it will take each data item A in the invalid read column and scan the data written column upward, beginning from the former record of DA_Table, to find the last updated value of A, which must have the correct value. Then, this value will be added as a pair (A, v) to the valid read column, and data item A will be deleted from the invalid read column and so on until all the data items in the invalid read column in the same record have been recovered. The second step

is to recompute each data item in the data written column in the same record by reading the new values from the valid read column. Certainly, after those two steps are successfully completed, all the data items in that record should have the correct values, as no attacking has occurred. The last step is to check the last column in the same record, which is the column (fog_ID), to find out if any data item from that record has been read by another fog node in the system; if so, a new valid data items table (VIT_Fog$_x$) will be created for each affected fog node if it does not exist yet. Then, the transaction ID along with the correct new values of each accessed data item will be added to VIT_Fog$x$. As soon as all the records in the damage audit table have been examined and all three steps are checked, the VIT_Fog$_x$ table will be sent to the corresponding fog node Fog$_x$. Finally, in steps 14-18, the new log that has just been generated while the data recovery algorithm is in process will be checked to make sure all the data items have correct and valid values.

4.6.2 Algorithm 4: Secondary Fog Node Data Recovery Algorithm

This algorithm is also similar to the main data recovery algorithm, Algorithm 3, excluding two main differences. The first is the input of Algorithm 4 will be three tables two of them come from Algorithm 2 for the same fog node which they are the damage audit table DA_Table and the damage item table (DIT_Fog$_x$). And the third table input is the new valid data items table (VIT_Fog$_x$) which comes from the data recovery algorithm of another fog node. Secondly, this algorithm will check every record on the received damaged audit table. When an affected transaction is found, then every data item A in the invalid column will be deleted after copying the (A, v) pair from the corresponding transaction on the VIT_Fog$_x$ table to the valid data column on DA_Table. After all the damaged data items have been

Algorithm 4.3 The Main Recovery Algorithm

```
1: for each record in the DA_Table do
2:   if invalid read column of T_i ≠ ∅ then
3:     for every data item A in invalid read column do
4:       find the last updated (A, v) pair in data written column of DA_Table from the former records
5:       add (A, v) to valid read column
6:       delete A from invalid read column
7:     end for
8:     for every A in data written column do
9:       recalculate the value of A using values in the valid read column
10:    end for
11:    if any fog_x is existing in fog_ID column then
12:      if VIT_ Fog_x does not exist then
13:        Create a new VIT_ Fog_x where x is the ID of aimed fog node that reads the affected transaction
14:      end if
15:      Add T_i and the (A, v) pair which is the correct value of A into VIT_ Fog_x
16:    end if
17:  end if
18: end for
19: Send VIT_ Fog_x to Fog_x node
20: for every A in DI_L do
21:   check the new log that has just been created while the recovery process was in progress
22:   if A is not modified in the log then
23:     scan data written column of DA_Table upward to find last updated value of A
24:     substitute the value of A in the database with v
25:   end if
26: end for
```

deleted on each record of the DA_Table, the data written column will be checked to discern if it is empty or not. If not, then the value v of each data item A in the data written column needs to be recalculated by using the new values in the valid read column. However, the same procedure used in Algorithm 3 will be employed if the record has a transaction with some data items on the invalid read column. The process continues until all the affected data items in the entire system are recovered to a consistent state.

Algorithm 4.4 Secondary Fog Node Recovery Algorithm

Once Fog_x receives the VIT_ Fog_x
for each record in the DA_Table **do**
  if T_i is affected transaction **then**
    for every data item A in invalid read column **do**
      copy the (A, v) pair from corresponding transaction on the VIT_ Fog_x to the valid data column on DA_Table
      delete A from invalid read column
    end for
    if the data written column $\neq \varnothing$ **then**
      for every A in data written column **do**
        recalculate the value of A using values in the valid read column
      end for
    end if
  else if T_i is updating transaction & invalid read column $\neq \varnothing$ **then**
    for every A in invalid read column **do**
      find the last updated (A, v) pair in data written column of DA_Table from the former records
      add (A, v) to valid read column
      delete A from invalid read column
    end for
    for every A in data written column **do**
      recalculate the value of A using values in the valid read column
    end for
    if any fog_y is existing in fog_ID column **then**
      if VIT_ Fog_y does not exist **then**
        Create a new VIT_ Fog_y where y is the ID of aimed fog node that reads the affected transaction
      end if
      Add T_i and the (A, v) pair which is the correct value of A into VIT_ Fog_y
    end if

    Send VIT_ Fog_y to Fog_y node
    for every A in DIT_ Fog_x **do**
      check the new log that has just been created while the recovery process was in progress
      if A is not modified in the log **then**
        scan data written column of DA_Table upward to find last updated value of A
        substitute the value of A in the database with v
      end if
    end for

4.7 An Example

To clarify the proposed scheme, let us consider the following example. Let us have in our smart city three fog nodes. Fog_1 and Fog_2 are public fog nodes to collect consumer data using smart meters. Fog_x is a private fog node for the utility company to manage aggregated data and calculate some important things, such as the bills and the amount of consumption. Consider the following log schedules for each one of them:

$\mathbf{S_{Fog.1}}$= $r_1(A,\ 5)\ r_1(B,\ 4)\ w_1(C,\ 11,\ 9)\ w_1(G,\ 3,\ 9)\ r_2(B,\ 4)c_1\ r_2(G,\ 9)\ w_2(A,\ 5,\ 13)\ w_2(D,\ 0,\ 13)c_2\ fog_2.r_3(G,\ 9)c_3\ w_4(A,\ 13,\ 5)\ w_4(G,\ 2,\ 3)c_4\ r_5(D,\ 13)\ r_5(A,\ 5)\ r_5(C,\ 9)\ w_5(D,\ 2,\ 27)c_5\ r_6(B,\ 4)\ w_6(B,\ 4,\ 4)\ r_6(D,\ 16)\ w_6(D,\ 16,\ 20)\ r_6(A,\ 5)\ w_6(A,\ 5,\ 25)c_6\ fog_2.r_7(D,\ 20)c_7\ r_8(\text{C},\ 9)c_8\ w_9\ (\text{C},\ 9,\ 11)c_9\ r_{10}(A,\ 25)\ r_{10}(C,\ 11)\ w_{10}(E,\ 10,\ 36)c_{10}\ fog_2.r_{11}(E,\ 36)c_{11}$

$\mathbf{S_{Fog.2}}$ = $r_9(K,\ 3)\ r_9(fog_1.T_3.G,\ 9)\ w_9(K,\ 3,\ 12)\ c_9\ r_{10}(M,\ 10)\ r_{10}(K,\ 12)\ w_{10}(M,\ 10,\ 22)c_{10}\ fog_x.r_{11}(M,\ 22)\ c_{11}\ \ldots\ r_{14}(fog_1.T_7.D,\ 20)\ r_{14}(L,\ 4)\ w_{14}(N,\ 17,\ 24)\ c_{14}\ r_{16}(fog_1.T_{11}.E,\ 36)\ w_{16}(P,\ 4\ ,\ 36)c_{16}\ fog_x.r_{17}\ (P,\ 36)\ c_{17}$

$\mathbf{S_{Fog.x}}$ = $r_8(O,\ 6)\ w_8(F,\ 6,\ 7)c_8\ r_9(fog_2.T_{11}.M,\ 22)\ r_9(H,\ 10)\ w_9(I,\ 14,\ 32)c_9\ w_{10}(J,\ 0,\ 18)\ c_{10}\ r_{11}(fog_2.T_{17}.P,\ 36)\ r_{11}(J,\ 18)\ w_{11}(Q,\ 60,\ 54)\ c_{11}\ r_{12}(Q,\ 54)\ r_{12}(U,\ 12)\ w_{12}(R,\ 57,\ 66)c_{12}$

Now, suppose the IDS detects the first transaction T_1 on the Fog_1 schedule as an attacking transaction and data items (C) and (G) as being maliciously updated. Therefore, the IDS will send the detected transaction T_1 as the list (MT_L) to Fog_1. Once Fog_1 receives the list, it will launch Algorithm 1 since it is the main victim fog node in the system. Then, it will create a new damage audit table (DA_Table) and a new damage items list (DI_L). Consequently, the log file of Fog_1 will be scanned, starting from the first attacking transaction

on the MT_L, which is T_1. Whenever an attacking transaction is found, such as T_1 in our example, it will be added to the DA_Table as a new record. All the write operations of T_1 will also be checked, so whenever a data item is found, it will be added along with its old "before image" value as a pair to the data written column, and the data items will be added to the damaged items list if they are not already there. In our example, the pairs (C, 11), (G, 3) are added to the data written column (see Table 4.2) while the data items (C) and (G) will be added to DI_L, and this will be the case for all attacking transactions that belong to (MT_L).

The algorithm will take the next transaction in the log file, which is T_2. Since T_2 is an updating transaction, it will be added into the DA_Table. Consequently, every reading operation in T_2 will be examined to ascertain if it reads any damaged data item from the DI_L; if so, the data item will be added to the invalid read column, as apparent with (G). Otherwise, it will be added as a pair with its value to the valid read column as (B, 4) in our example. The next transaction is T_3, as we notice that Fog_2 has read the data item (G), which is a damaged data item. Steps 9-16 will examine that and mark it in the fog ID column. Also, it will create a new damage item table for Fog_2 (DIT_Fog_2) and add it to the table with the transaction ID and the timestamp of the transaction when it has been committed (see Table 4.3). Steps (28-30) of the algorithm will find that the damaged data items (A) and (G) have been refreshed in T_4 because they updated without reading any other damaged data items. Therefore, they will be added along with their new values in the data that were written and removed from the DI_L. The process continues until the end of the log. Then, the DIT_Fog_2 will be sent to Fog_2 to be used as input for Algorithm 2, while the DA_Table and DI_L will be sent to Algorithm 3, which is the main recovery algorithm.

Once Fog_2 receives the DIT_Fog_2, it will launch Algorithm 2 and use DIT_Fog_2 as input to do further assessment and detection. Note that this table will add any damaged data items that are detected in Fog_2, as shown in Table 4.6. Thus, a new damage audit table DA_Table will be created and initialized to null. It will scan the local log of Fog_2, starting from the first affected transaction on the DIT_Fog_2, which is T_9 in our example. Therefore, T_9 will be added to the new DA_Table and for every read operation, the data item will be examined to find out if it belongs to the DA_Table; if so, it will be added to the invalid read column. Otherwise, it will be added to the valid read column along with its value. Therefore, the data item ($fog_1.T_3.G$) will be added to the invalid read, while the pair (K, 3) will be added to the valid read column (see Table 4.4). However, it will do the same thing for the write operations as in Algorithm 1, so the updated data item (K) along with its value (K, 12) will be added to the data written column. Meanwhile, the data item (K) will be added with its timestamp into the DIT_Fog_2, as can be seen in Table 4.6, since it becomes affected by reading the damaged data item ($Fog_1.T_3.G$). For T_{10}, the updating occurs after reading the damaged data item (K), and the access time of T_{10} for that data item (K) comes after the timestamp of the damage data item in the DIT_Fog_2, so the scenario (the same as shown in T_2 in Fog_1) will be repeated, as shown in Table 4.4. Note that examining the timestamp of reading the damage data item is important in Algorithm 2 since Fog_2 may read the same data item twice from Fog_1, once before it became damaged and once after that, so any item accessed before the damage must be valid. Also, the damaged data item may be refreshed after it has been read by Fog_2. Therefore, the only way to determine the status is to compare the committed times. Consequently, the process continues until the end of the log. By the end, there will be two tables, the DA_Table for fog_2, which will be

sent to Algorithm 4 to conduct the data recovery, and the new damage item table for Fog_x (DIT_Fog_x) (Table 4.5), which will be sent to Fog_x to be used as an input for Algorithm 2 to do further detection. Once Fog_x receives the DIT_Fog_x, the same process as we discussed previously will be continued until the end of the log of Fog_x (see Tables 4.7 and 4.8).

As soon as Fog_1 has done the damage assessment algorithm (Algorithm 1) and sent the DA_Table and DI_L to Algorithm 3 to do the data recovery, which will be launched immediately, and taken the DA_Table and DI_L as inputs, it will scan the DA_Table from the start and search for any transactions that read invalid data items. For example, T_2 reads invalid data item (G), and then the algorithm will look for the last valid update value of (G), which must be the closest transaction before the T_2. Therefore, T_1 has to have the latest updated correct value of (G), which is (3). Therefore, the pair (G, 3) will be copied to the valid read column, and (G) will be removed from the invalid read column. After that, T_2 will be recalculated using the new values, as apparent in Table 4.9. Note that in this example, only the addition operation will be used for simplicity; therefore, we consider that for any transaction where a write operation is found after read operations, all the values of the read operations will be added together. After that, the T_3 will be taken, so it will do the same process. Additionally, it will find that Fog_2 has read the damage data item (G). Hence, a new valid data item table for Fog_2 (VIT_Fog_2) will be created and added to the transaction ID, which is T_3 and the correct value of (G), which is (G, 3) (see Table 4.10), and so on until the end of the DA_Table. After that, VIT_Fog_2 will be sent to Fog_2 to be used as an input in the Algorithm 4 to recover the data.

Once Fog_2 receives the VIT_Fog_2, it will launch Algorithm 4 and use VIT_Fog_2 along with it is own DA_Table to do data recovery. Then, every record in the DA_Table will be

checked. Since the first record (T_9 in our example) must be an affected transaction from Fog_1, then VIT_Fog_2 should have the correct and valid value of the damaged data item. Therefore, the new value of data item (G) will be copied from VIT_Fog_2 to the valid read column of the DA_Table and remove it from invalid read. After that, T_9 will be recalculated using the new values (as shown in Table 4.11). The rest of the algorithm will be the same as Algorithm 3, and the same thing will be done in Fog_x after it receives the VIT_Fog_x from Fog_2 (see Tables 4.12 and 4.13).

Table 4.2: The Damage Audit Table for Fog_1

T Id	**Data written**	**Valid read**	**Invalid**	**Fog ID**
T_1	(C, 11), (G, 3)			
T_2	(A, 13), (D, 13)	(B, 4)	G	
T_3			G	Fog_2
T_4	(A, 5), (G, 3)			
T_5	(D, 27)	(A, 5)	C, D	
T_6	(D, 20), (A, 25)	(B, 4), (A, 5)	D	
T_7			D	Fog_2
T_9	(C, 11)			
T_{10}	(E, 36)	(C, 11)	A	
T_{11}			E	Fog_2

Table 4.3: Fog_2 Damage Item Table Created by Fog_1

Transaction Id	**Damaged Data Items**	**Timestamp**
$fog_1.T_3$	G	9:00:00 AM
$fog_1.T_7$	D	9:00:30 AM
$fog_1.T_{11}$	E	9:01:00 AM

Table 4.4: The Damage Audit Table for fog_2

T Id	Data written	Valid read	Invalid	Fog ID
T_9	(K, 12)	(K, 3)	$fog_1.T_3.G$	
T_{10}	(M, 22)	(M, 10)	K	
T_{11}			M	Fog_x
T_{14}	(N, 24)	(L, 4)	$fog_1.T_7.D$	
T_{16}	(P, 36)		$fog_1.T_{11}.E$	
T_{17}			P	Fog_x

Table 4.5: Fog_x Damage Item Table Created by Fog_2

Transaction Id	Damaged Data Items	Timestamp
$fog_2.T_{11}$	M	9:00:20 AM
$fog_2.T_{17}$	P	9:01:05 AM

Table 4.6: DIT_Fog_2 with all damaged data items that are found on Fog_2

Transaction Id	Damaged Data Items	Timestamp
$fog_1.T_3$	G	9:00:00 AM
$fog_1.T_7$	D	9:00:30 AM
$fog_1.T_{11}$	E	9:01:00 AM
T_9	K	9:00:00 AM
T_{10}	M	9:00:17 AM
T_{14}	N	9:00:30 AM
T_{16}	P	9:01:00 AM

Table 4.7: The Damage Audit Table for fog_x

T Id	Data Written	Valid Read	Invalid	fog_{ID}
T_9	(I,32)	(H,10)	$fog_2.T_{11}.M$	
T_{11}	(Q,54)	(J,18)	$fog_2.T_{17}.P$	
T_{12}	(R,66)	(U,12)	Q	

Table 4.8: DIT_Fog_x with all damaged data items that are found on Fog_x

Transaction Id	Damaged Data Items	Timestamp
$fog_2.T_{11}$	M	9:00:20 AM
$fog_2.T_{17}$	P	9:01:05 AM
T_9	I	9:00:20 AM
T_{11}	Q	9:01:08 AM
T_{12}	R	9:01:10 AM

Table 4.9: DA_Table for fog_1 after damaged data have been recovered

T Id	Data written	Valid read	Invalid	Fog ID
T_1	(C, 11), (G, 3)			
T_2	(A, 7), (D, 7)	(B, 4), (G, 3)		
T_3		(G, 3)		Fog_2
T_4	(A, 5), (G, 3)			
T_5	(D, 23)	(A,5), (C, 11), (D, 7)		
T_6	(D, 27), (A, 32)	(B, 4), (A, 5), (D, 23)		
T_7		(D, 27)		Fog_2
T_9	(C, 11)			
T_{10}	(E, 43)	(C, 11), (A, 32)		
T_{11}		(E, 43)		Fog_2

Table 4.10: VIT_Fog_x sent from fog_1

Transaction Id	Valid Data Items
$fog_1.T_3$	(G,3)
$fog_1.T_7$	(D,27)
$fog_1.T_{11}$	(E,43)

Table 4.11: DA_Table for fog_2 after damaged data have been recovered

T Id	Data Written	Valid Read	Invalid	fog_{ID}
T_9	(K, 6)	(K,3), ($fog_1.T_3$.G, 3)		
T_{10}	(M, 16)	(M, 10), (K, 6)		
T_{11}		(M, 16)		Fog_x
T_{14}	(N, 31)	(L, 4), ($fog_1.T_7$.D, 27)		
T_{16}	(P, 43)	($fog_1.T_{11}$.E, 43)		
T_{17}		(P, 43)		Fog_x

Table 4.12: VIT_Fog_x sent from fog_2

Transaction Id	Valid Data Items
$fog_2.T_{11}$	(M,16)
$fog_1.T_{17}$	(P,43)

Table 4.13: DA_Table for fog_x after damaged data have been recovered

T Id	Data Written	Valid Read	Invalid	fog_{ID}
T_9	(I, 26)	(H, 3), (fog_2.T_{11}.M, 16)		
T_{11}	(Q, 61)	(J, 18), (fog_2.T_{17}.P, 43)		
T_{12}	(R, 73)	(U, 12), (Q, 61)		

4.8 Experiments and Evaluation

In section 3.6 in chapter 3, we explained the experimental setup, the simulation environment, and datasets that have been used in detail.

4.8.1 Evaluation of the Second Model: Ensuring Integrity in Smart City Model

This model proposes the use of fog computing in smart cities to manage utility service companies and consumer data. We also propose a novel technique to assess damage to data caused by a malicious attack. The original data can then be recovered, and the database returned to a consistent state as though no attack has occurred. This experiment aimed to analyze the behavior of the proposed damage assessment algorithm as it detected affected data items with varying factors designed for the experiment. As declared in the simulation section, different sets of log files were produced. Each set represented the number of fog nodes in the system with a different number of transactions in the log files. In this experiment we started with a fixed number of fog nodes each time and a different number of transactions in each log file. Each time a malicious transaction was arbitrarily injected, the results were categorized into different sets of affected transactions: a set of 20 to 25 affected transactions, a second set of 10 to 15 identified, affected transactions, and a third set of less than five identified, affected transactions. The clustering of the affected transactions was necessary

in evaluating the proposed algorithms as they are 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 twenty times. Then the total time, from insertion of the malicious transaction until all damage audit tables for all affected transactions were built, was computed. Then the average for each set to identify the damaged data items was calculated. Also, the total time from receipt of the damage audit table for the proposed data recovery algorithms to recover all affected data items was computed. Then the time required for each set for the recovery process was averaged. This average was calculated for use in investigating and evaluating our approach. We compared the results, determining the factor or factors having a greater or lesser impact to the algorithms.

4.8.1.1 First Experiment: The impact of log file size on different number of fog nodes

The relationship between the number of transactions in each log file, the number of affected transactions, and the average time required by our system to build all damage audit tables for all affected data items in the whole system were plotted on Figs. 4.2, 4.3, 4.4, and 4.5. The average time was calculated by averaging the detection time each set required to complete the tasks. The results show that a log file of 100 transactions will take the system approximately 60 to 120 ms to detect all affected data items in the whole system no matter the number of affected transactions. Also, the figures demonstrate that the delay time for the log files of 100 transactions increased by 15 to 20 ms from the set of less than five affected transactions to the set of 10 to 15 affected transactions. And the delay time the log files of 100 transactions increased by 20 to 40 ms from the set of 10 to 15 affected transactions to

the set of 20 to 25 affected transactions.

The time required to detect all affected data items and create the audit tables for all affected transactions in the whole system with log files of 500 and 1000 transactions was increased from the log files of 100 transactions for all different sets. Therefore, the time needed for the system to identify all affected data items in log files of 500 transactions was about 165 to 185 ms for the set of less than five affected transactions. The time required for the same processes increased by an average of 26 ms for the set of 10-15 affected transactions and by an average of 76 ms from the set of 10 -15 affected transactions to the set of 20 to 25 affected transactions. Additionally, the delay time for the system to complete the damage assessment algorithms in log files of 1000 transactions was roughly 190 ms for the set of less than five affected transactions. Then it increased by an average of 130 ms for the set of 10-15 affected transactions which was the greatest shifting in all results. Nonetheless, it increased a bit, by an average of 26 ms, from the set of 10 -15 affected transactions to the set of 20 to 25 affected transactions. This increase in delay time is related to the number of transactions in the log file, the position of the malicious transaction in the log file, and the number of fog nodes infected. For instance, Fig. 4.5 notes that the average delay time for the set of 10-15 affected transactions on 20 fog nodes with 500 log file transactions is considerably greater than its counterpart in the other groups of fog nodes. Analysis of the results found that the malicious transactions were randomly inserted in this particular set to the beginning of the log files, leading to many more affected fog nodes which, in turn, led to this delay. Therefore, it follows that the main reasons for the delay results in the other experiments are determined by the variables introduced: the number of affected fog nodes, or the location of the malicious transactions in the log file(s), or the number of affected transactions. The

latter of which will impact system availability.

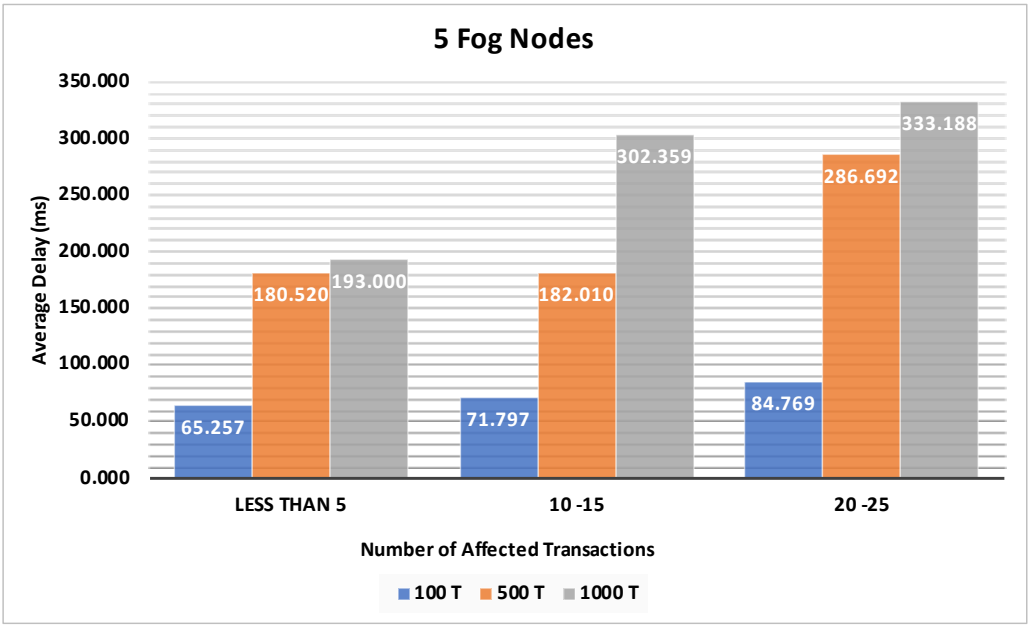

Figure 4.2: The impact of log file size on five fog nodes in the smart city model.

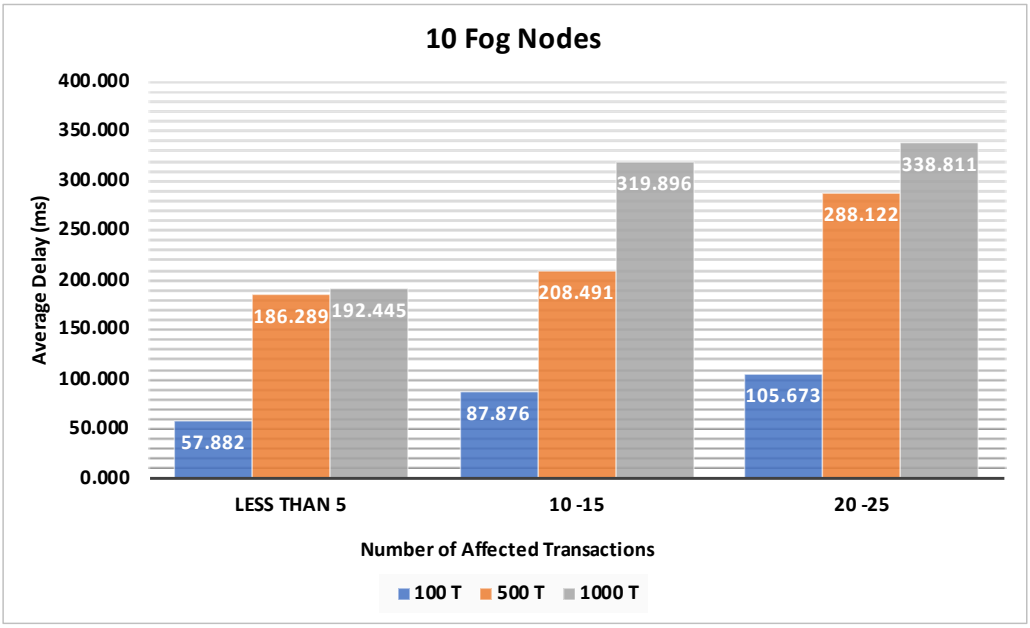

Figure 4.3: The impact of log file size on ten fog nodes in the smart city model.

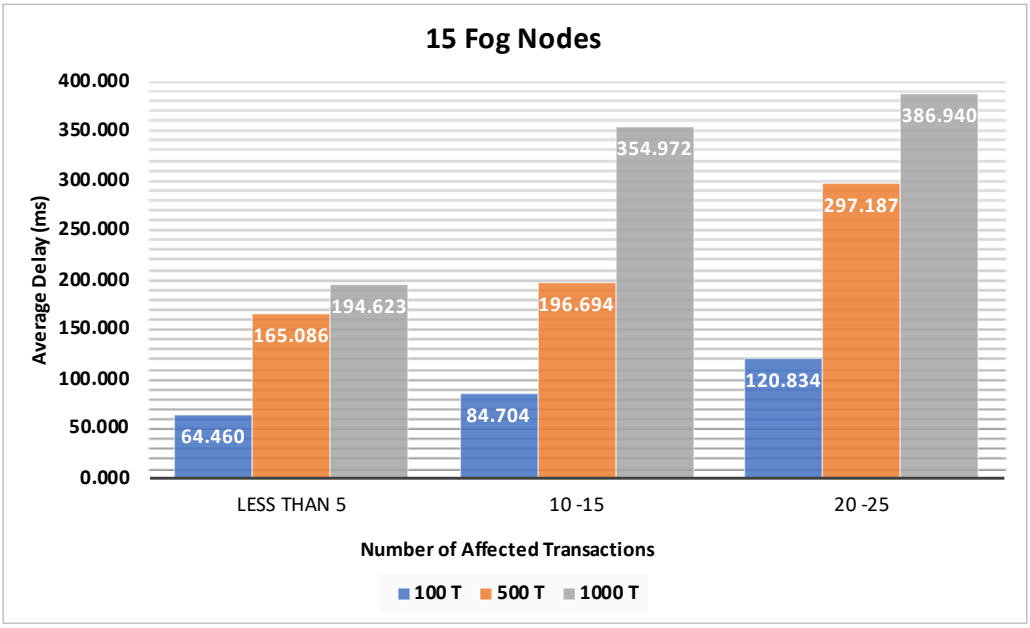

Figure 4.4: The impact of log file size on fifteen fog nodes in the smart city model.

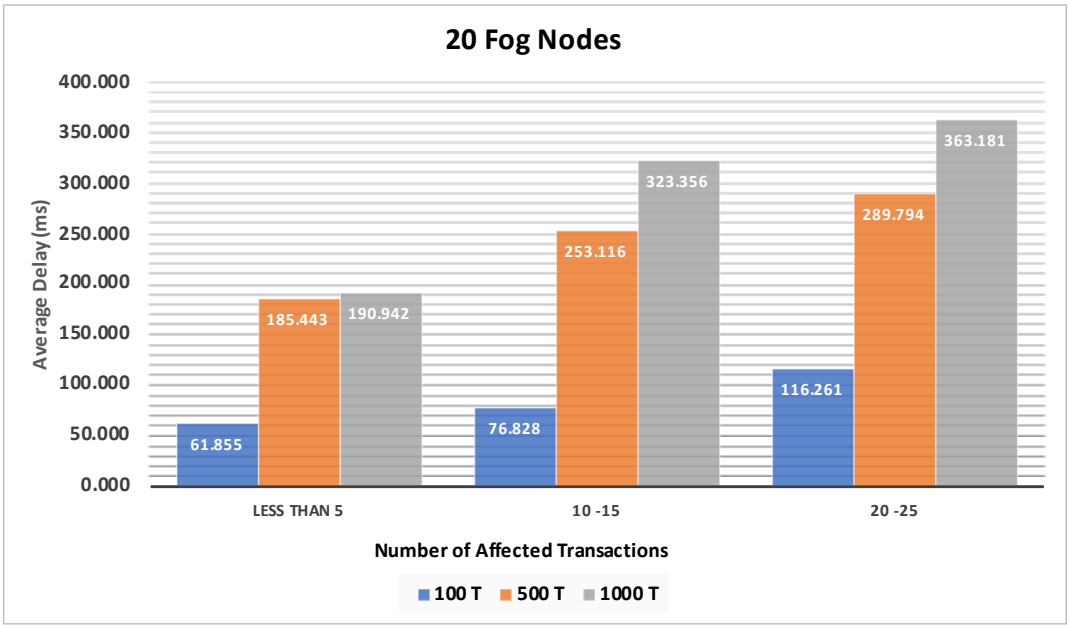

Figure 4.5: The impact of log file size on twenty fog nodes in the smart city model.

4.8.1.2 Second Experiment: The impact of a variable number of fog nodes and sets of affected transactions on log files with different log files size

This experiment was designed to study the impact of a variable number of fog nodes on our damage assessment algorithm. Hence, the relationship between the number of fog nodes and the sets of affected transactions have been illuminated in this experiment. The experiment was conducted each time with log file sizes fixed to sets of 100 transactions, then to 500 transactions, then to 1000 transactions. Figs. 4.6, 4.7, and 4.8 illustrate steady performance by the proposed damage assessment algorithms with the set of less than five affected transactions regardless of log file size.

However, the time the damage assessment algorithm needed to complete its tasks increased when the number of fog nodes increased with the sets of 10 to 15 affected transactions and 20 to 25 affected transactions. This held true regardless of log file size. The increase in latency occurred here as the rising number of fog nodes on the system gave way to the chance of a greater number of infected fog nodes, especially with many affected transactions.

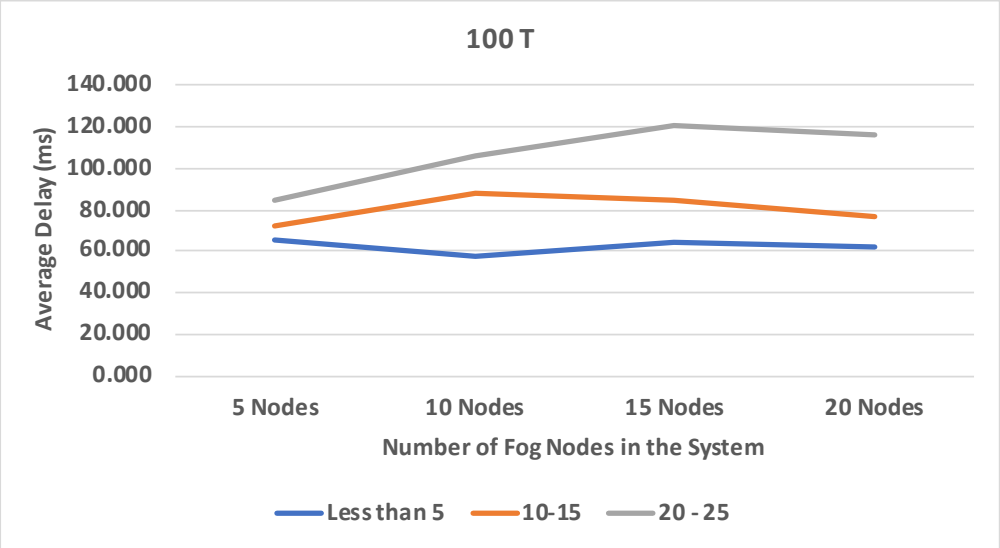

Figure 4.6: The impact of a variable number of fog nodes and sets of affected transactions on log files of 100 T in the smart city model.

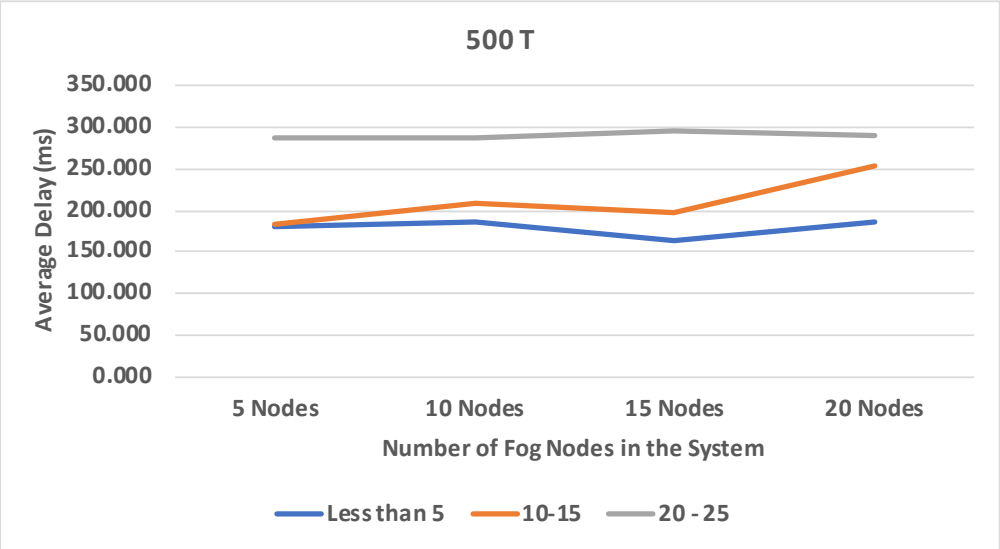

Figure 4.7: The impact of a variable number of fog nodes and sets of affected transactions on log files of 500 T in the smart city model.

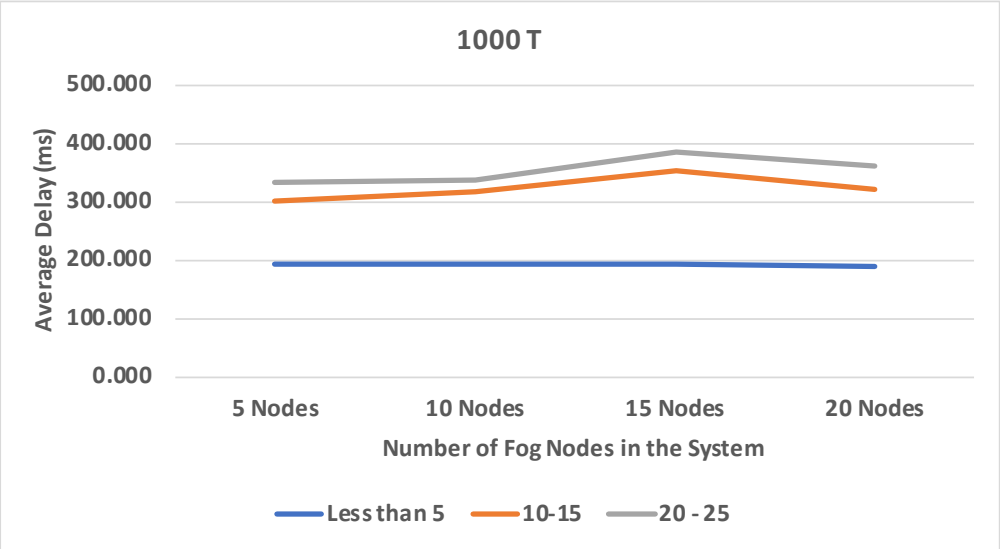

Figure 4.8: The impact of a variable number of fog nodes and sets of affected transactions on log files of 1000 T in the smart city model.

4.8.1.3 Third Experiment "The Evaluation of the Proposed Data Recovery Algorithms"

This experiment measures the performance of our proposed data recovery algorithms in regaining all damaged data items and restoring their correct values as if no attack occurred. For each affected fog node, we calculated the time required, once the damage audit table and damage item list has been received, for our proposed data recovery algorithms to recover and render all the affected data items to a consistent state. Then the average time for each set of affected transactions that we mentioned in Experiments 4.8.1 was computed. The following Figs. 4.9, 4.10, 4.11, and 4.12. show the relationship between different numbers of transactions on each log file and variable sets of affected transactions with a fixed number of fog nodes.

Overall, we notice that the essential time to recover all damaged data items in the

whole system necessitates no more than 2.8 ms in all cases with all different numbers of fog nodes, sets of affected transactions, and sizes of log files. The set of less than five affected transactions needs approximately 1.1 to 1.5 ms to recover all data items regardless of the number of fog nodes and the log file size; the other two sets took 0.09 to 0.96 ms more time. Hence, during this time the affected data items will be unavailable for use. It is impacted by the number of affected transactions in the system and holds true regardless of the number of fog nodes or the number of transactions in each log file.

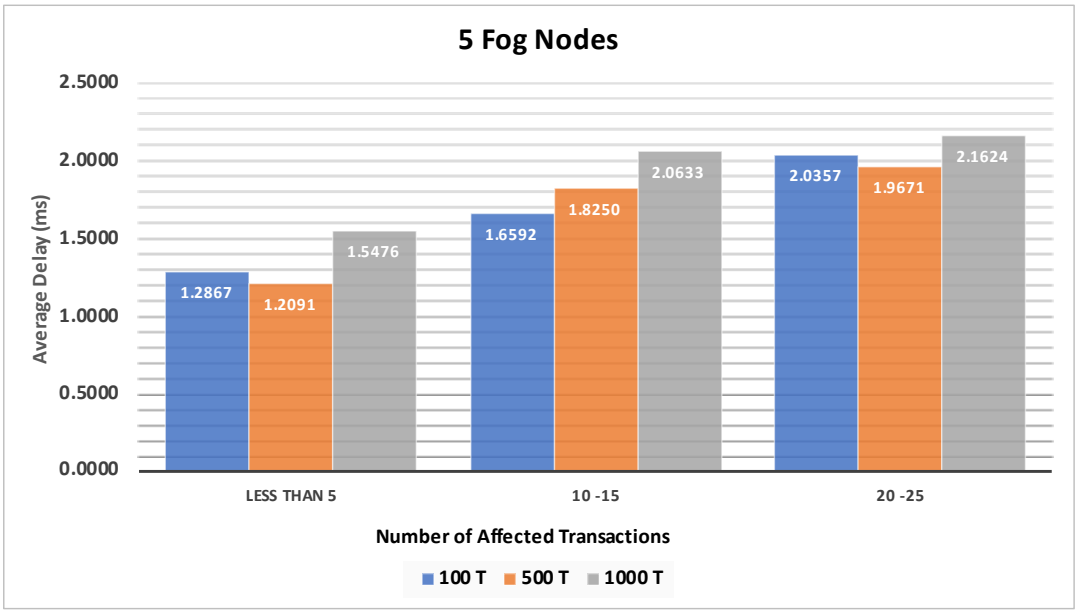

Figure 4.9: The impact of log file size on five fog nodes in the smart city model.

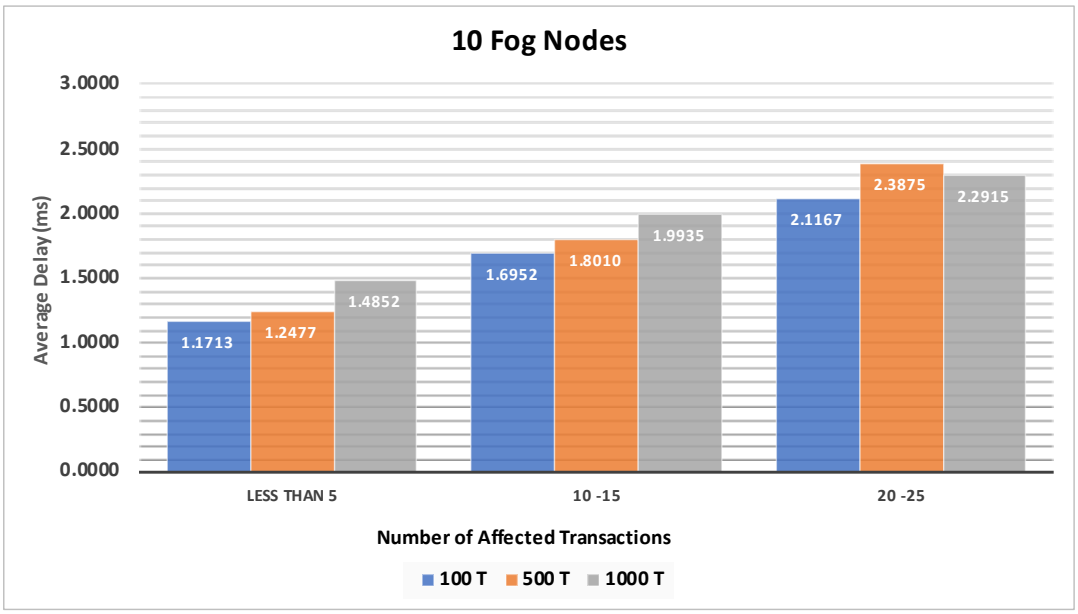

Figure 4.10: The impact of log file size on ten fog nodes in the smart city model.

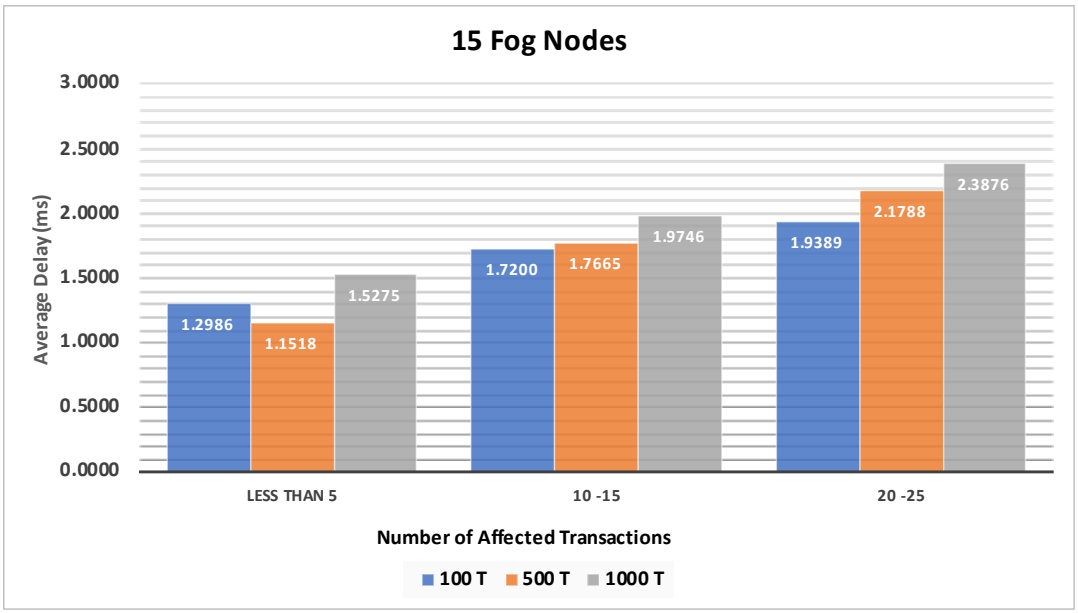

Figure 4.11: The impact of log file size on fifteen fog nodes in the smart city model.

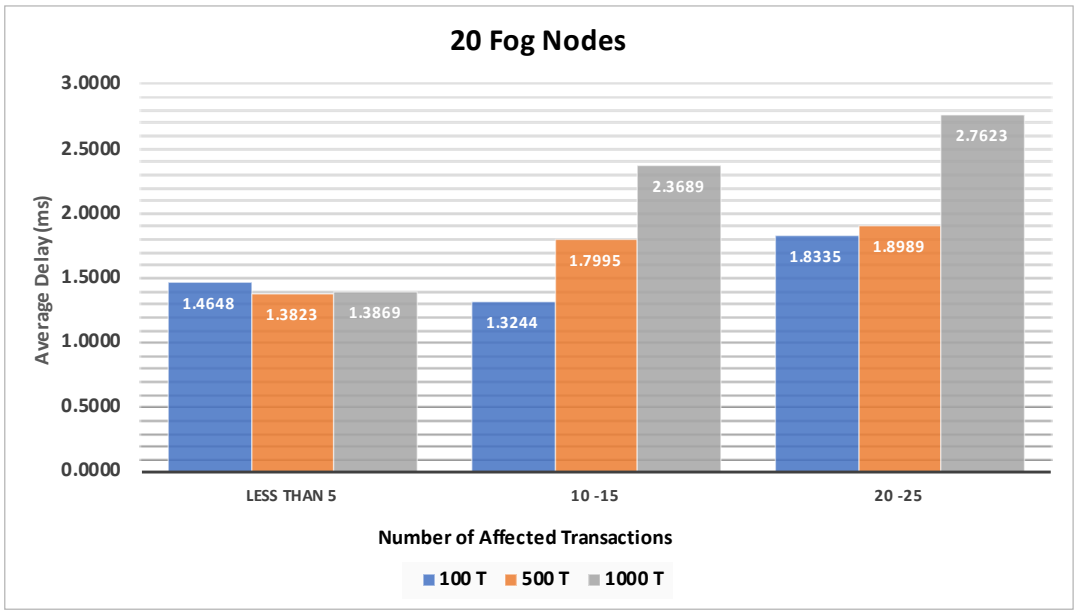

Figure 4.12: The impact of log file size on twenty fog nodes in the smart city model.

5 Ensuring Data Integrity in Fog Nodes Distribution in Intelligent Government Model

Some of the following paragraphs, figures, and algorithms that will be introduced in this chapter have been already published in our work [64, 81] as shown in the publications and reprint permissions chapter 7.

5.1 Introduction

In this era of technological evolution, many technologies, including the Internet of Things (IoT), wearable devices, smart buildings, video gaming and streaming, and smartphones, are an integral part of daily human life. Building stable and reliable infrastructure for IoT systems necessitates consideration of rapid growth in the number of connected IoT devices. It is estimated that these devices will number approximately 75 billion by 2025 [1] and generate more than 79 zettabytes of data daily [2]. The present cloud and internet architecture face significant challenges in handling and managing this massive amount of data [5].

Another challenge present cloud and internet architecture faces is the provision of adequate security and privacy measures for end users' sensitive data. Government data is of particular concern as attack on data within a government system may place a nation at risk [25]. Sensitive applications such as traffic control systems, Real-Time Surveillance Camera Monitoring, and Video-conference applications require real-time processing and location awareness.

To overcome the limitations of cloud computing, fog computing, as a new platform, was introduced by Cisco Systems in January, 2014. Fog computing is a virtualization architecture in which vital computing resources are re-located close to the data source and the end-user. Fog computing is able to process massive amounts of data, extend cloud storage, and offers rapid local responses [7]. Fog computing services make it a very satisfactory and practical design solution for many sensitive applications that require real-time, location awareness, including those systems on every level of government. Fog computing strengthens privacy and security of data as data is stored and computed close to end-users at the edge of the network [6, 9].

Governments around the world are creating intelligent environment systems that will improve the quality of services they provide and the living conditions of their citizens. By utilizing fog computing technology, they will be able to capture and utilize maximum benefit of their systems and provide a safe environment with high-quality services. However, these technologies do not come without risks. One of the most crucial issues in fog computing is preserving the security and privacy of consumer data. While several researchers have attempted to address security issues in intelligent government systems [82], there remain some issues in need of more attention, including that of assessing damage to data that suffer malicious attacks.

Data damage assessment and recovery are fundamental to creating secure and reliable databases. This is particularly true for a government data environment where the transmission of time-critical sensitive data is commonplace. If one considers, for example, traffic control systems, Real-Time Surveillance Camera Monitoring applications, as mentioned earlier, any attack on these systems will paralyze their vital functions such as real-time processing

and location awareness features. When an attack on these critical databases violate integrity of stored data the incident will result in serious consequences including damage to property and even loss of life. Therefore, in addition to development of appropriate mechanisms for adaptation of fog computing in intelligent systems, meeting security requirements to respond to attacks by providing fast and accurate damage assessment and recovery techniques are absolutely necessary.

Thus the primary objectives of this section are twofold: propose a novel model design for an intelligent government system using fog computing technology to control and manage the data across the entire system and propose a unique scheme to detect and assess data affected by malicious attacks in the model system.

5.2 Model

This section proposes using a unique distributed fog computing architecture to develop an intelligent county government system for managing important data. It also presents a cooperative technique for identifying and assessing damaged and affected data following an attack. In the proposed model, the Intrusion Detection System (IDS) is responsible for detecting malicious transactions in the system and providing a copy of those transactions to the damage-assessment algorithms. Each fog node in the proposed architecture 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.

5.3 Model Notation

A description of the notations to be used in our model can be found in Table 5.1.

Table 5.1: Notation used in our model

Notation	Description
sub-system	Distributed fog nodes sub-system representing each authority/government agency.
pr_fog	Primary fog data service node on the sub-system.
tsnt_fog	Transient fog nodes responsible for collecting data by using smart devices from end users across the county.
S{pr_fog$_{ID}$.T$_M$}	The set of detected malicious transactions in the system detected by IDS.
G(T$_n$, E)	Graph Representation where T$_n$ denotes the maximum number of transactions in the node and E denotes the number of edges
Aff-Lfog$_n$	Affected-transactions list containing all affected transactions identified by proposed damage assessment algorithms.
$w_i(A, v_1, v_2)$	The write operation of the transaction T$_i$; v_1 is the before image, which represents the value of the data item A before updating. And v_2 is the after image, which is the value of data item A after it is updated.
$r_i(A, v)$	The read operation of transaction T$_i$ where A is the data item and v is the current value of A.
c_i	The transaction T$_i$ has been committed, which means it is successfully completed and will be recorded to the database.

5.4 The Proposed Architecture

Generally, there are two main types of fog computing technology, public fog computing, and private fog computing. Any distributed fog computing network to which access is limited and restricted for use only by specific parties shall be a private fog computing network. A great example of a private fog computing network is the campus fog computing distribution such as school or college, and government fog computing distribution as what we will have here in this chapter as an example [83].

In this proposed model, each government agency operates its own fog nodes subsystem. Each sub-system has at least one primary fog-data-service node (pr_fog) and multiple transient-fog nodes (tsnt_fog), which are well distributed to ensure quality of service

throughout the county. Some sub-systems may have restricted access to other sub-systems.

The tsnt_fog are responsible for collecting data using IoT and smart devices, security surveillance cameras, and kiosk stations from end-users across the county. Then the data is transferred to the pr_fog based on several factors, including location and selective data access restrictions. Each pr_fog in the system is able to perform some imperative operations such as calculation and aggregation of required data. These operations are necessary to optimize the bandwidth of networks since the data sent through it are aggregated by force. The aggregation of essential data increases privacy and security because most of the data is now processed locally at the edge of the network; local processing minimizes the transmission of sensitive data over the network.

For example, the police department has one pr_fog and several tsnt_fogs that are responsible for controlling and managing traffic lights, while other tsnt_fogs are responsible for collecting data from security cameras and so on. All data collected by traffic-light sensors and surveillance cameras are transferred through tsnt_fogs to the police-department pr_fog. Likewise, certain other pr_fogs in the system, such as the fire department's pr_fog, have restricted access to the data collected by traffic-lights sensors; the fire department's access allows emergency vehicles to change traffic lights and admission to the least congested routes.

As we stated early this model is a private distributed fog computing network so, It is assumed that the whole system can be trusted because it is operated and surveilled by the government. However, there is one trusted fog node in the system managing security matters such as key management and distribution. It is located at the center of the system, ensuring a reliable connection to all subsystems and providing a secondary pathway in the event a pr_fog is disconnected. The trusted fog node plays a leading role in damage assessment and

data recovery process (Fig. 5.5).

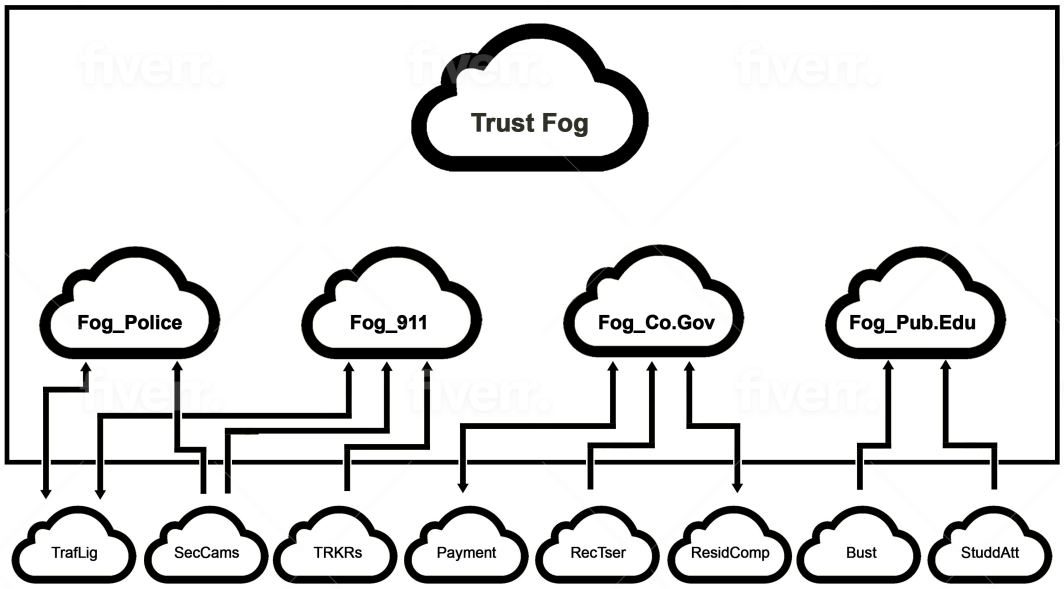

Figure 5.1: The Proposed Architecture.

5.5 The Proposed Damage Assessment Algorithms

This model implements a transaction-dependency graph to observe and monitor all transactions in the entire system. Transaction-dependency graphs allow quick detection of all transactions affected by a malicious transaction. Transaction dependency is used instead of a complicated data dependency where transaction updates are difficult to track and graphs are sparser and more isolated. A transaction-dependency graph is easier to build and capable of detecting and blocking malicious transactions significantly faster, thus preventing further damage to data in the system. Unaffected data items are available for immediate use. The

following subsections explain in detail each step of the proposed model.

5.5.1 Algorithm 5.1: Building Local Graphs

Each primary fog-data-service node pr_fog builds 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 pr_fog generates a graph by building a two-row matrix. The first row contains associated data items and the second row contains the last transaction that updated each of those data items (Tables 5.2 and 5.3). When a new transaction T_i is committed to the database, the matrix is scanned to determine T_i's precursors (parents). The new transaction is added as a new vertex to the graph and the edges between T_i and its parents are inserted based on those dependencies. Coincidentally, the matrix is updated with the newly committed transaction. So, for each write operation for data item X in T_i, the algorithm will scan the matrix and update the last transaction for data item X to T_i. However, if data item X does not exist in the matrix, it will insert that data item and the transaction identification T_i as a new record.

For example, consider the following log records for pr_fog_x and pr_fog_y:

pr_fog_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 $w_4(A, 48, 69)$ $w_4(G, 18, 24)$ c_4 $r_5(D, 35)$ $r_5(A, 69)$ $w_5(D, 35, 98)$ c_5 $r_6(B, 34)$ $w_6(B, 34, 63)$ $r_6(D, 98)$ $w_6(D, 98, 24)$ $r_6(A, 69)$ $w_6(A, 69, 78)$ c_6 $r_7(D, 24)$ c_7 r_8(C, 80)c_8 $r_9(fog_y.T_6.V, 56)$ 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}.

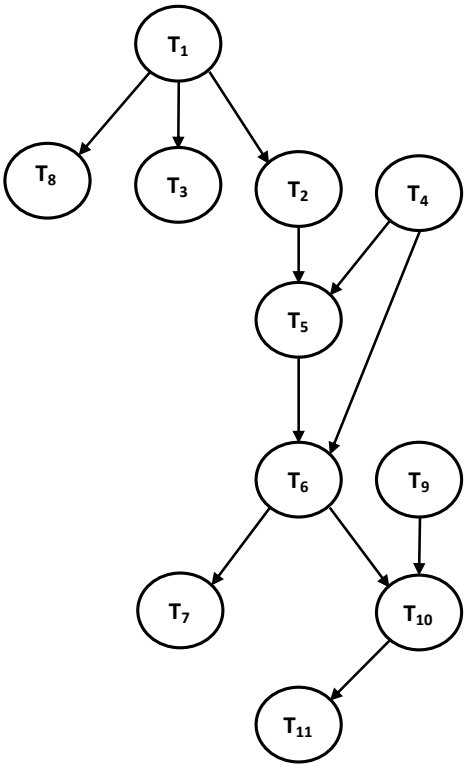

Figure 5.2: Described Local Dependency Graphs for pr_fog$_x$.

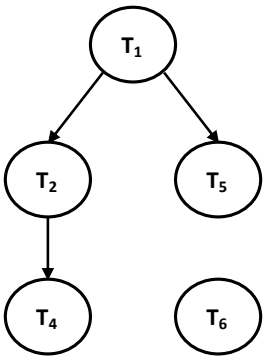

Figure 5.3: Described Local Dependency Graphs for pr_fog$_y$.

pr_fog$_y$: $r_1(K,\ 58)\ r_1(fog_x.T_4.G,\ 24) w_1(K,\ 58,\ 98) c_1\ r_2(M,\ 39)\ r_2(K,\ 98) w_2(M,\ 39,\ 59) c_2\ r_3\ (R,\ 43) c_3\ r_4(fog_x.T_5.D,\ 35)\ r_4(M,\ 59)\ w_4(N,\ 36,\ 84)\ c_4\ r_5(K,\ 98)\ w_5(P,\ 43,\ 78) c_5\ w_6(V,\ 56)\ c_6$

The log records for pr_fog$_x$ indicate that T_6 read data items B, D, and A. After T_6

Algorithm 5.1 Building Local Graphs

1: Create a new Hash-table H and Initialize to null
2: Create a new Graph Representation $G = (\mathrm{T}_n, E)$ and Initialize to null
3: When T_i is committed in the local DB then
4: **for** each operation $O \in \mathrm{T}_i$ **do**
5: **if** O is $w_i\ (A, v1, v2)$ **then**
6: add a pair (A, Ti ID) to H
7: add Ti 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_j ID from H
12: add edge from T_j to T_i
13: **else if** $A \in$ foreign fog node **then**
14: call global graph algorithm (Algorithm 5.2)
15: **end if**
16: **else if** O is c_i **then**
17: **end if**
18: **end for**=0

is committed, it is added as a new vertex to the local graph of $\mathrm{pr\_fog}_x$, and the directed-dependency edge (the link) between T_6 and other transactions on the graph is drawn based on T_6's dependency relationships.

The algorithm locates the last updated transactions in the matrix for data items B, D, and A (Table 5.2). 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 5.3). 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 5.3.

Table 5.2: 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 5.3: 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}

5.5.2 Algorithm 5.2: Building Global Graphs on the Trust Fog Node

As the trusted fog node plays an essential role in managing security and protection considerations, including damage assessment and data recovery, the node obtains global graphs for any transaction that accessed a data item in a pr_fog node in the system.

Here is an example:

- T_i belonging to pr_fog_y accesses data item A. The last transaction updating data item A was T_j on pr_fog_x.
- A copy of that last transaction's graph updating data item A is sent to the global graph on the trusted fog node and includes all preceding vertices that could affect T_j directly or indirectly.
- Transaction T_i is added to the global graph as a new child of T_j and an edge is drawn from transaction x.T_j to y.T_i.
- Transaction T_i from pr_fog_y continues updating the global graph on the trusted fog node by sending a copy of every subsequent transaction committed locally at pr_fog_y (successor of T_i). This copy will be updated and sent frequently.

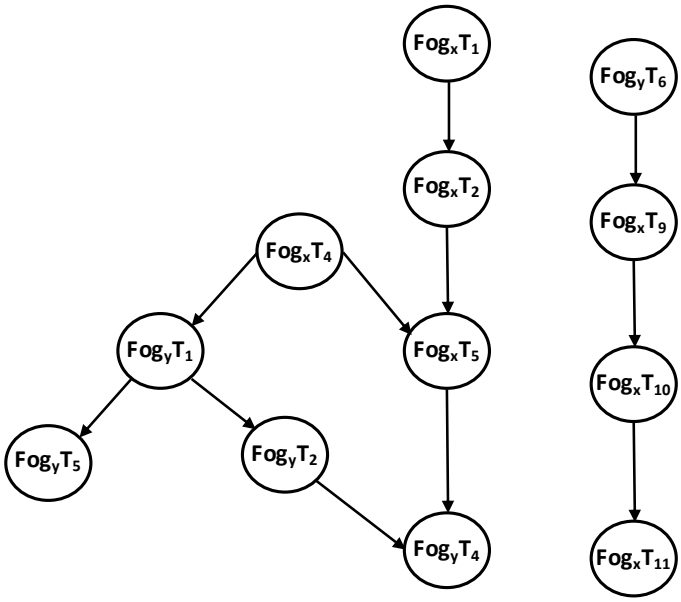

Figure 5.4: The Global Graph on The Trusted Node.

Figure 5.4, shows an example of the global graph generated using transaction activities at the trusted fog node from pr_fog$_x$ and pr_fog$_y$. As seen in the log records of pr_fog$_y$ T$_1$ read data item G, which was updated by T$_4$ in the log records of pr_fog$_x$. Then T$_4$, pr_fog$_x$ 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

Algorithm 5.2 Building Global Graphs on the Trusted Fog Node

1: Create a new Global Graph Representation G$_g$ = (T$_n$, E) on trusted fog node and Initialize to null
2: (When T$_i$ $\in$ fog$_x$ reads data item A $\in$ fog$_y$) then
3: **for** each r_i (A, v) $\in$ T$_i$ **do**
4: acquire all predecessors of T$_j$ from G$_y$
5: add Ti as a new vertex to G$_g$ if it does not exist
6: add edge from fog$_y$.T$_j$ to fog$_x$.T$_i$
7: **end for**
8: (When a new transaction T$_z$ $\in$ pr_fog$_x$ is committed where T$_z$ is successor of T$_i$) then
9: add Tz as a new vertex to G$_g$ if it does not exist
10: add edge from T$_i$ to T$_z$

vertex since it does not have parents. In addition, once T1 from pr_fog$_y$ is committed, it is added to both the global graph on the trusted fog node and the local pr_fog$_y$ graph.

All new transactions added to the local graph of pr_fog$_y$ and the successor of T_1 are added to the global graph. Thus:

- In the log records of pr_fog$_y$, T_2 reads data item K from T_1. T_2 is added as a new vertex in the local pr_fog$_y$ 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 pr_fog$_y$ 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 pr_fog$_y$ 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.

5.5.3 Algorithm 5.3: Damage Assessment Algorithms

Once the IDS detects a malicious transaction in the system, it sends it to the trusted fog node for identifying all transactions dependent on, and therefore affected by, the malicious

transaction. Simultaneously, each victimized primary fog pr_fog data service node in the system receives the malicious transaction(s) detected on its database.

The pr_fog can scan its own graphs and detect affected transactions. This feature is necessary as some local graphs containing transactions, malicious or not, have never been accessed or read by another fog node. Those transactions have not been forwarded to the trusted fog node. The pr_fog scans only the graphs that it possesses using a modified Depth First Search (MDFS) algorithm, beginning with the malicious transaction received from the IDS.

Immediately following this part of the damage-assessment process, each subsystem receives a list of all affected transactions detected on its local database. All data items updated by those transactions, hence considered damaged, are blocked from being accessed until recovered. The rest of the data items are made available to users.

Algorithm 5.3 Damage Assessment Algorithm

1: Once S{pr_fog$_{ID}$.T$_M$} is received from IDS
2: Classify S{pr_fog$_{ID}$.T$_M$} to multi-set {S$_1${pr_fog$_1$.T$_M$}..S$_n${pr_fog$_n$.T$_M$}}
3: Send each subset S$_n${pr_fog$_n$.T$_M$} to the (pr_fog$_n$)
4: Send S{pr_fog$_{ID}$.T$_M$} to the trust fog node.
5: For the trust fog node and each other pr_fog$_n$ that receive S or S$_n$.
6: MDFS (G$_n$, T$_{M1}$)
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$_{M1}$ to tranS
10: mark T$_{M1}$ as visited
11: **while** tranS is not empty **do**
12: T$_{M1}$ = tranS.pop
13: add T$_{M1}$ to Aff-Lfog$_n$
14: **for** each child T$_i$ of T$_{M1}$ 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**

5.6 Applying the Transaction-Dependency Graph Scheme to public distributed fog computing network

The implementation of the transaction-dependency graph scheme can also be applied to the previous model architecture; the architecture for the use of a distributed fog node system in smart cities to manage the consumer data of utility services. As we observed, this model lacks the availability of a trusted fog node to build global graphs and handle security matters such as damage assessment and data recovery. This model has only public fog nodes distribution and belongs to more than one owner. No trusted fog node can exist in this kind of distribution.

However, we can still modify our transaction-dependency graph scheme to enable observation and monitoring of all transactions in the entire system of a model lacking a

trusted fog node. As we mentioned in section 4.2, Ensuring Integrity in Smart City Models, each fog node in the proposed architecture must have its own log file and use a strict serializable history and the graphs produced must be stored in a separate, unmodifiable file with the original transactions in the same log file directory. These three assumptions remain compulsory for building local graphs for each fog node.

5.6.1 Algorithm 5.4: Building Local Graphs for public distributed fog nodes

Each public fog node (pub_fog) or private fog node for a utility service company (usc_fog) builds a directed graph based on transaction dependency relationships between transactions that have been successfully committed to its database. However, 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 on the basis of the dependencies. The fog node will update the tracking mechanism in accordance with the recently committed transaction.

To illustrate the algorithm, we use the following example for the records of log for pr_fog_1 and pub_fog_2 fog nodes:

pr_fog$_1$: $r_1(X,\ 32)\ w_1(Z,\ 29,\ 80)\ w_1(W,\ 89,\ 81)\ c_1\ r_2(Z,\ 80)\ w_2(X,\ 32,\ 22)\ w_2(D,\ 12,\ 35)\ c_2\ r_3(W,\ 81)$ *r3(*$\text{pub\_fog}_2.T_2.F,\ 66)\ w_3(W,\ 81,\ 91)\ c_3\ w_4(X,\ 22,\ 77)\ c_4 r_5(D,\ 35)\ r_5($ $X,\ 77)\ w_5(D,\ 35,\ 98)\ c_5 r_6(D,\ 98)\ w_6(D,\ 98,\ 24)\ r_6(X,\ 77)\ w_6(X,\ 77,\ 78)\ c_6$

pub_fog$_2$: $r_1(E,\ 13)$ $r_1(\text{pr\_fog}_1.T_1.W,\ 81)\ w_1(\ E,\ 13,\ 39)\ c_1\ r_2(F,\ 53)\ r_2(E,\ 39)\ w_2($ $F,\ 53,\ 66)\ c_2\ r_3(R,\ 43)\ c_3\ r_4(\text{pr\_fog}_1.T_5.D,\ 98)\ r_4(F,\ 66)\ w_4(J,\ 79,\ 30)\ c_4\ r_5(E,\ 39)\ w_5(I,$ $43,\ 78)\ c_5$

The log record for pr_fog_1 indicates that T5 reads data items (D) and (X). Conse-

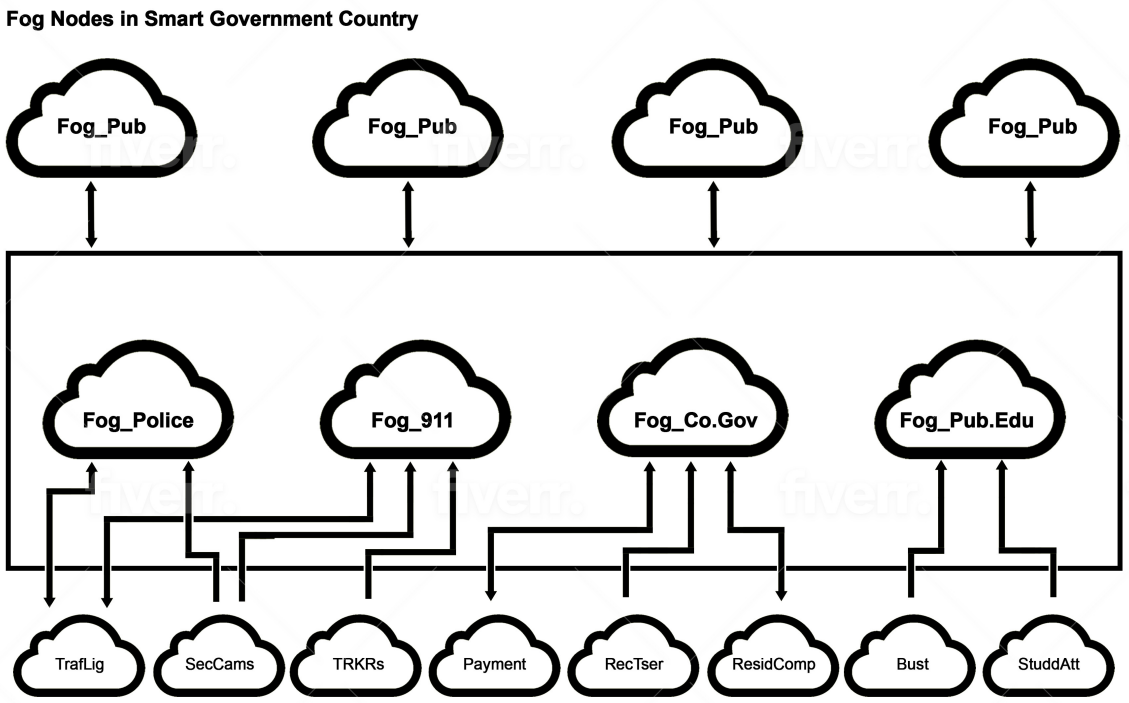

Figure 5.5: The Proposed Architecture.

quently, after T_5 is committed, it will be added as a new vertex to the local graph of pr_fog_1, 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 5.6).

However, when any transaction T_i reads a data item in one pub_fog / pr_fog node that has been updated by T_j in another pub_fog / pr_fog node, then the transaction identification (ID) of T_i will be added as a child of T_j on the local graph of the pub_fog / pr_fog node, in which the item is initially updated. For example, in the log records of pub_fog_2, T1 reads

Algorithm 5.4 Building Local Graphs for Public Distributed Fog Computing

```
1: Create a new Hash-table H and Initialize to null
2: Create a new Graph Representation G = (T_n, E) and Initialize to null
3: When T_i is committed in the local DB then
4: for each operation O ∈ 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) && A ∈ H then
9:       add T_i as a new vertex to G if it does not exist
10:      acquire the parents T_j ID from H
11:      add edge from T_j to T_i
12:   else if T_k ∈ foreign fog node fog_y reads data item A that updated by T_i then
13:      add fog_y.T_k as a new vertex to G if it does not exist
14:      add edge from T_i to fog_y.T_k
15:   end if
16: end for
```

data item W that is updated by T_1 in the log records of pr_fog_1. The transaction ID of T_1 from pub_fog_2 will then be added as a foreign child of T_1 on the local graph of pr_fog_1. T_4 from the log of pub_fog_2 reads item D that is written by T_5 on pr_fog_1, and the transaction ID of T_4 from pub_fog_2 will be added as a foreign child of T_5 on the local graph of pr_fog_1.

Data item F is updated by T_2 on pub_fog_2 and accessed by T_3 from pr_fog_1. Accordingly, T_3 will be added as a foreign child of T_2 on the local graph of pub_fog_2. The process continues updating local graphs, given that transactions are successfully committed to the local database. Figure 5.6 shows examples of the local graphs generated using transaction activities at pr_fog_1 and pub_fog_2.

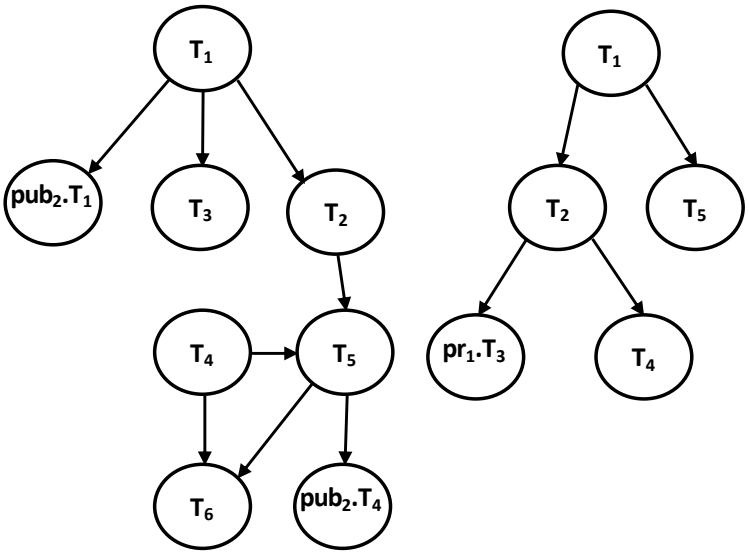

Figure 5.6: Described Local Dependency Graphs for pr_fog$_1$ and pub_fog$_2$.

5.6.2 Algorithm 5.5: Damage Assessment Algorithms for Public Distributed Fog Computing

Once the IDS detects a malicious transaction in the system, each fog node, pr_fog or pub_fog is informed of the malicious transaction(s) detected on its database. The pr_fog or pub_fog scrutinizes its own graphs and identifies affected transactions. The compromised fog nodes examine only the graphs that they possess using a modified Depth First Search algorithm (MDFS), 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_i from pub_fog$_2$ is found to be a successor of the initial malicious transaction T_j on the pr_fog$_1$, then a sub-list of affected transaction will be sent to pub_fog$_2$ 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

Algorithm 5.5 Damage Assessment Algorithm for Public Distributed Fog Computing

```
Once S{fog_ID.T_M} is received from IDS
Assort S{fog_ID.T_M} to multi-set S_1 ... S_n
Send S_x{fog_x.T_M} to the (fog_x)
For each fog_x that receive S_x
MDFS (G_n , T_M1 )
Establish a new list for affected transactions (Aff-Lfog_x)
Establish a new Stack S_T
add T_M1 to S_T
mark T_M1 as visited
while S_T is not empty do
    T_M1 = S_T.pop
    add T_M1 to Aff-Lfog_x
    for each child T_i of T_M1 in G_n do
        if T_i ∉ Aff-Lfog_x then
            add T_i to S_T
            mark T_i as visited
            add T_i to Aff-Lfog_x
            if T_i is foreign transaction ∈ fog_y then
                add T_i to sub-list Aff-Lfog_x,y
            end if
        end if
    end for
end while
Send Aff-Lfog_x,y to fog_y to do further detection
```

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.

5.7 Experiments and Evaluation

In section 3.6 in chapter 3, we explained the experimental setup, the simulation environment, and datasets that have been used in detail.

5.7.1 Evaluation of the Third Model: Private Fog Nodes Distribution in Intelligent Government County Model

In this experiment, we aimed to determine if the trusted fog node in the system which is proposed to manage security matters such as key management and distribution, could play a leading role in damage assessment. In this model we studied the behavior of the damage assessment algorithm as it detected affected transactions with differing factors.

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 comprises less than five identified, affected transactions. The second set contains ten to fifteen identified, affected transactions, and the third set includes thirty to thirty-five identified, affected transactions. This clustering is important for making a fair and reasonable comparison of the proposed algorithms as they are impacted by other factors such as the number of fog nodes and the number of transactions in each log files. Each transaction in each set was repeated approximately twenty 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.

5.7.1.1 First Experiment: The impact of the different sets of affected transactions on various fog nodes number

Figures 5.7, 5.8, 5.9, and 5.10 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. And this applies in all cases, regardless of the number of fog nodes or the number of transactions on each log file.

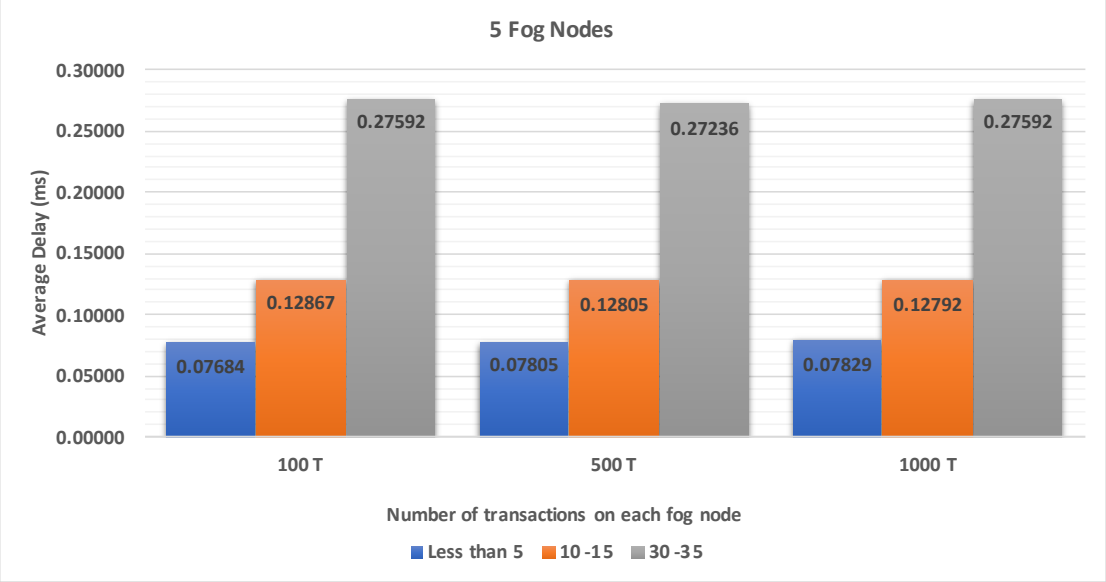

Figure 5.7: The impact of the different sets of affected transactions on five fog nodes

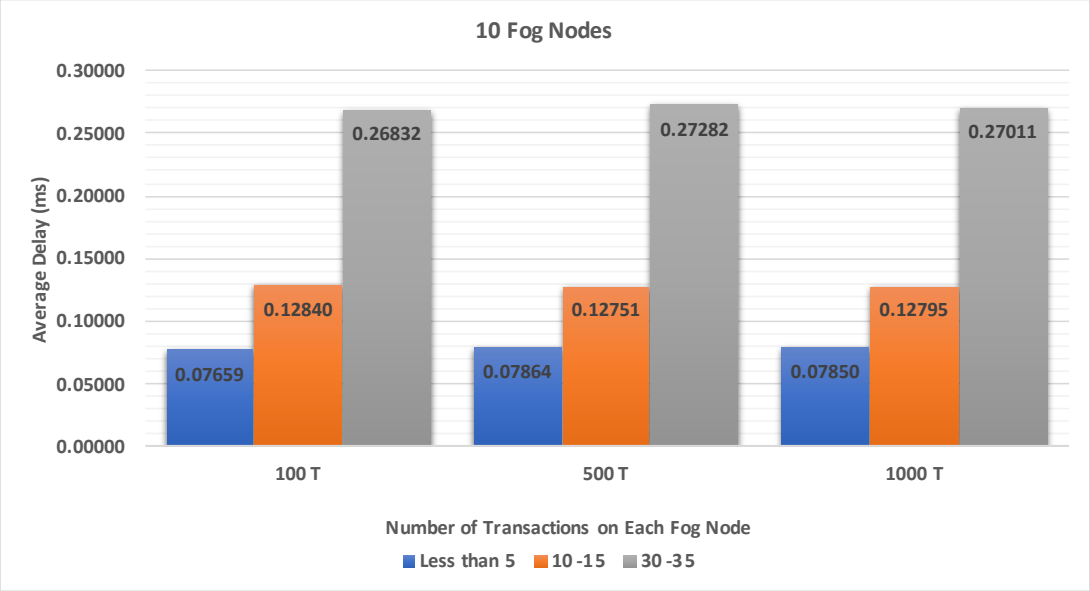

Figure 5.8: The impact of the different sets of affected transactions on ten fog nodes

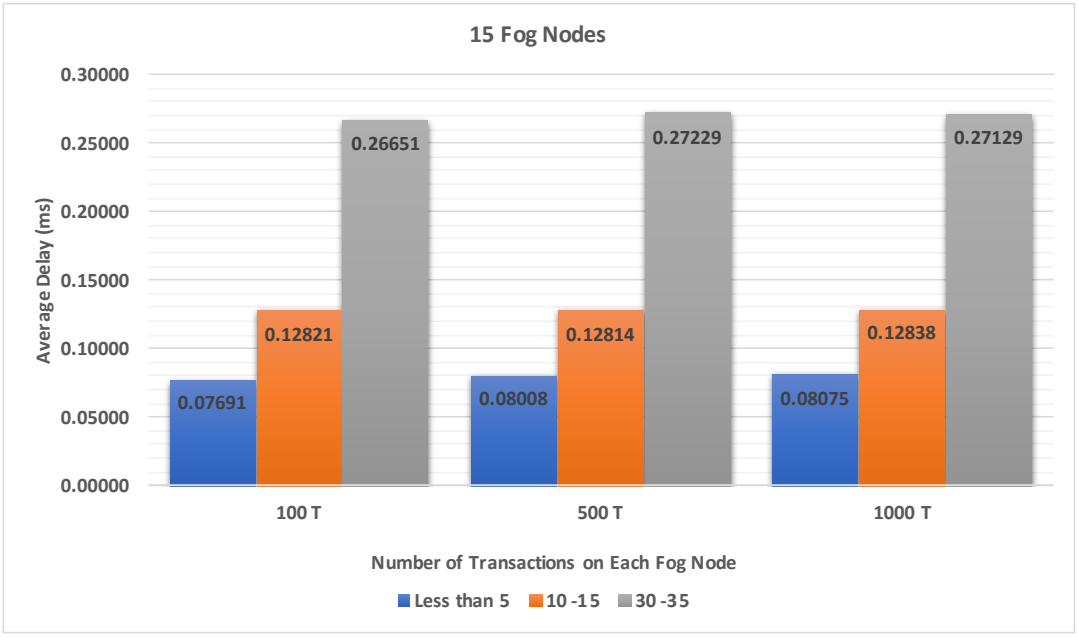

Figure 5.9: The impact of the different sets of affected transactions on fifteen fog nodes

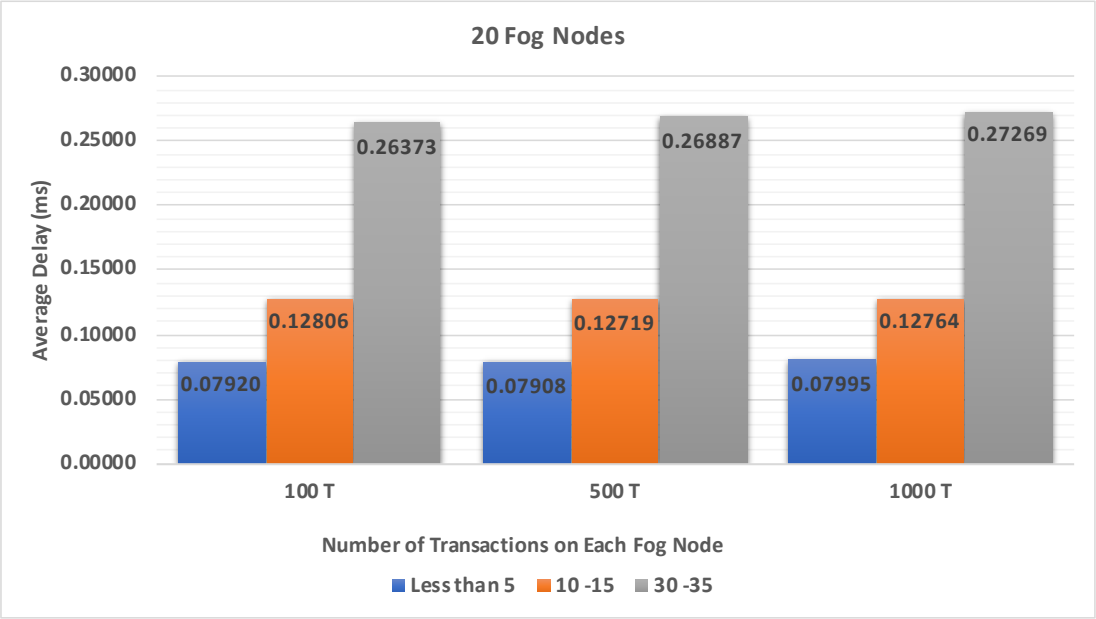

Figure 5.10: The impact of the different set of affected transactions on twenty fog nodes

5.7.1.2 Second Experiment: The impact of different number of transactions on various number of fog nodes

Figures 5.11, 5.12, 5.13, and 5.14 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. And 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.

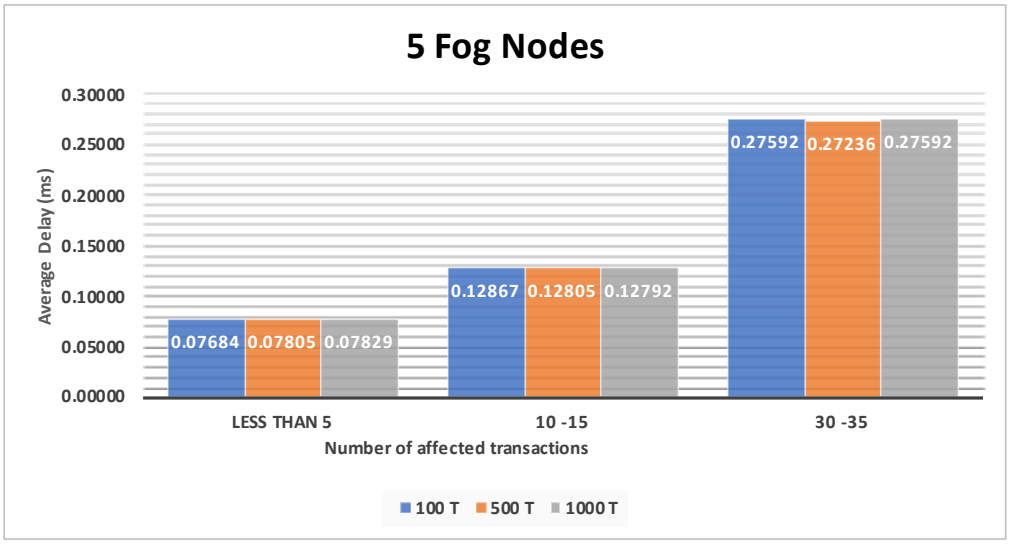

Figure 5.11: The impact of different number of transactions on five fog nodes

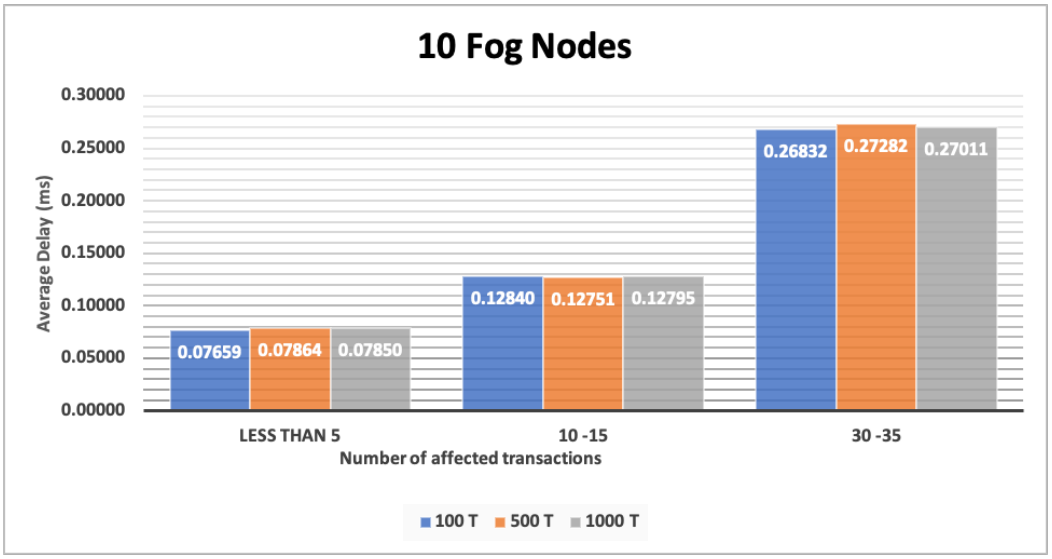

Figure 5.12: The impact of different number of transactions on ten fog nodes

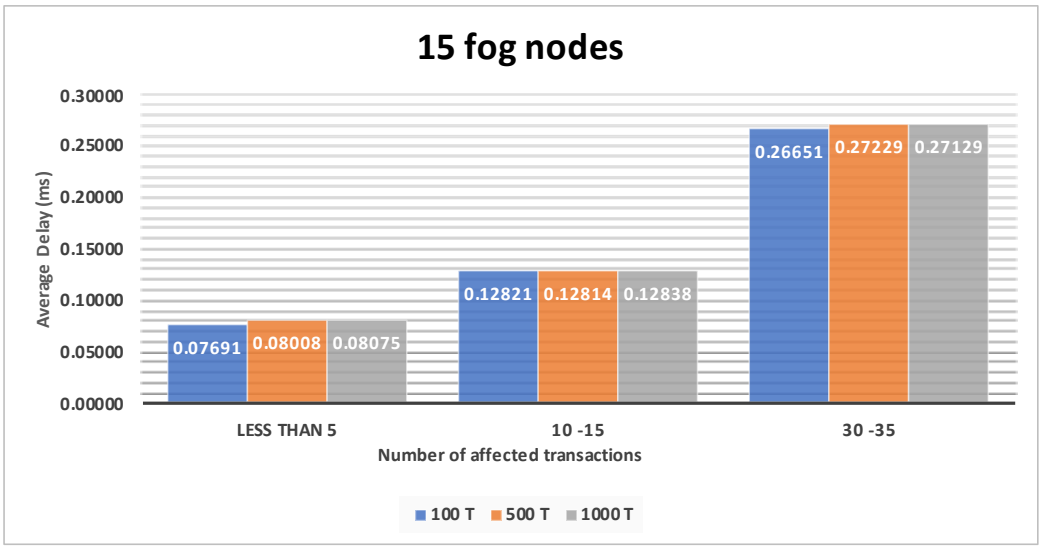

Figure 5.13: The impact of different number of transactions on fifteen fog nodes

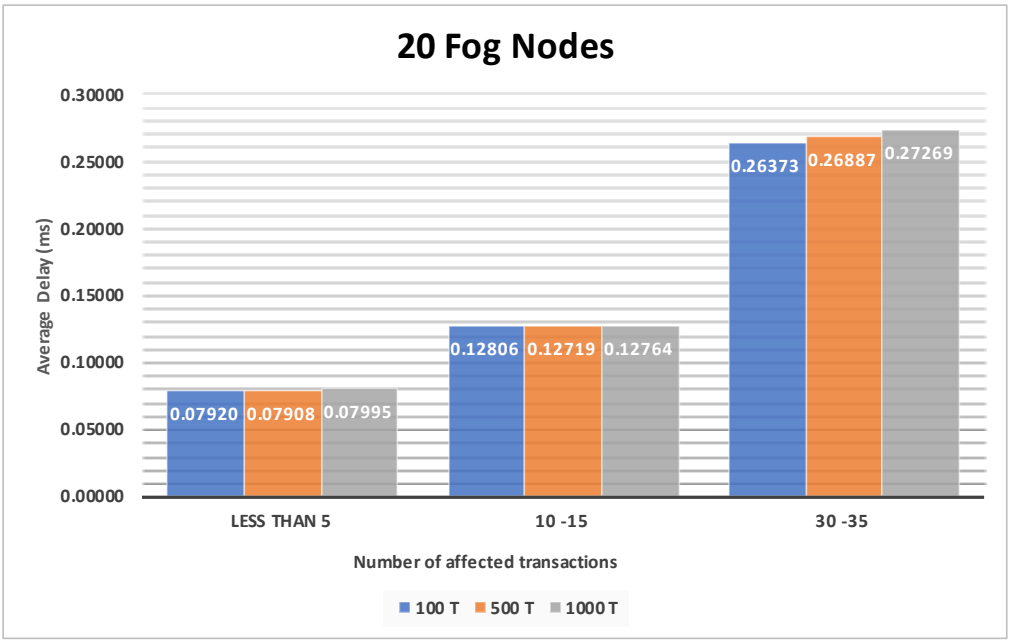

Figure 5.14: The impact of different number of transactions on twenty fog nodes

5.7.1.3 Third Experiment: The impact of a different number of transactions and fog nodes on different sets of affected transactions

This experiment represents the performance of our detection algorithm in identifying all affected transactions. The following Figures. 5.15, 5.16 and 5.17 show the relationship between the number of transactions on each log file and the number of fog nodes. The sets of affected transactions that were used in this experiment were fixed at less than five, then ten to fifteen, and finally to the set of thirty to thirty five.

Figure 5.15 indicates the set of affected transactions was fixed at less than five and the time needed for our damage assessment algorithm to detect all affected transactions was slightly increased when the number of fog nodes was increased. This scenario held true regardless of log file size. The delay that occurred here was due to increasing size in the global graph. The global graph size usually expands with a rise in transaction dependency, We observed the global graph increasing in size with both the number of transactions on each log file and the number of fog nodes. We also observed that the number of transactions in each log file had little impact on the results. Hence, the log files containing 1000 and 500 transactions took only slightly more time than those containing 100 transactions when we used the set of less than five affected transactions.

But when the set of affected transactions was fixed at 10 to 15, the results differed, as seen in Fig. 5.16. Here we see that an increase in the number of fog nodes decreased the delay time. With the larger set of affected transactions, the global and local graph levels with small log file size and fewer fog nodes show greater delay than the graphs with large log files and more fog nodes.

The set of 30 to 35 affected transactions produced different findings as shown on figure 5.17. A small log file of 100 transactions exhibited the same behavior as the previous set of 10 to 15 affected transactions where the smaller number of fog nodes showed more time delay. The delay decreased when the number of fog nodes increased. The log files containing 500 transactions were fairly stable for a bit before showing a decrease in delay with the increase in fog nodes to 20, minimizing the number of transaction (node) levels on the graphs. Although, for the 1000 transaction log file the time needed to detect all affected transactions increased slightly due to global and local graph growth.

To conclude, these experiments prove that the sooner IDS identifies the malicious transactions the more efficient and faster damage assessment of the affected transactions can be made. The large spread of affected transactions is the primary cause of delay in our algorithms. This is due to the rise in the height and depth of the affected sub-tree.

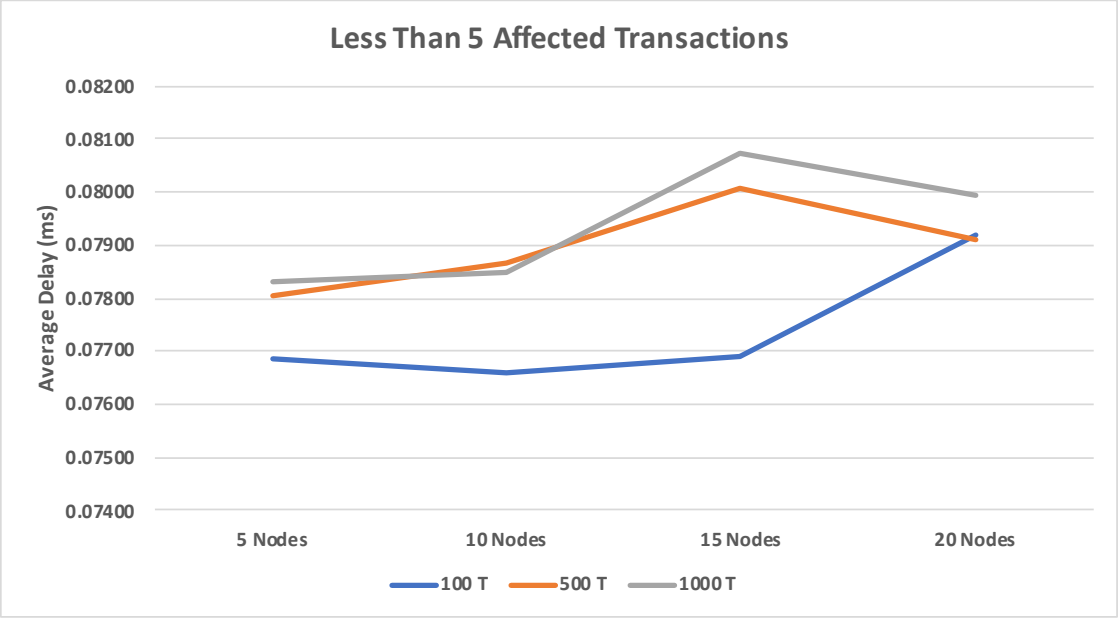

Figure 5.15: The impact of a different number of transactions and fog nodes on a set of less than five affected transactions

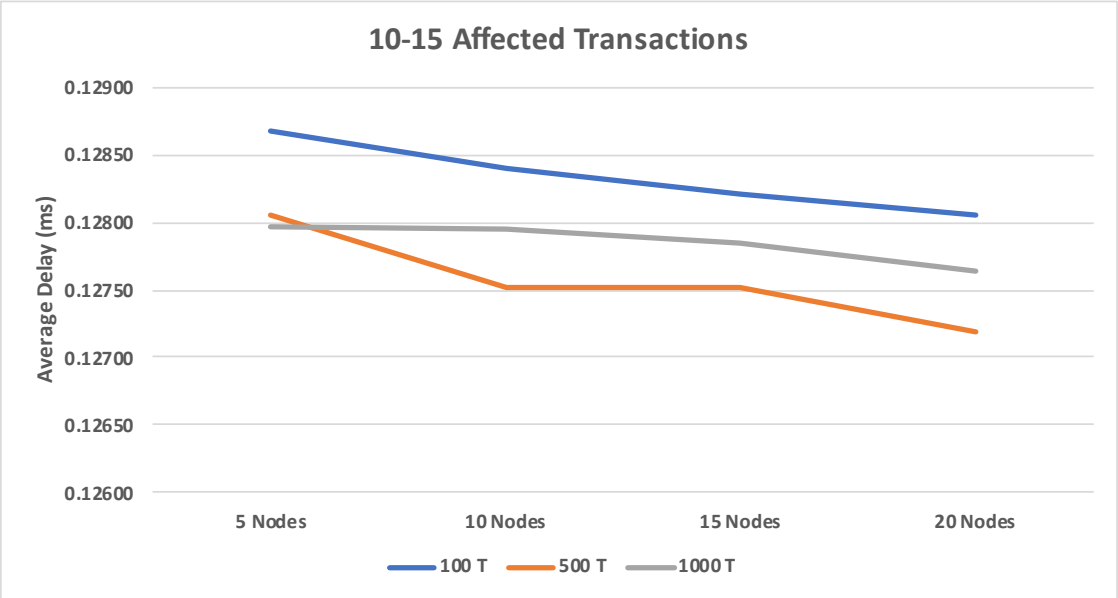

Figure 5.16: The impact of different number of transactions and fog nodes on set of ten to fifteen affected Transactions

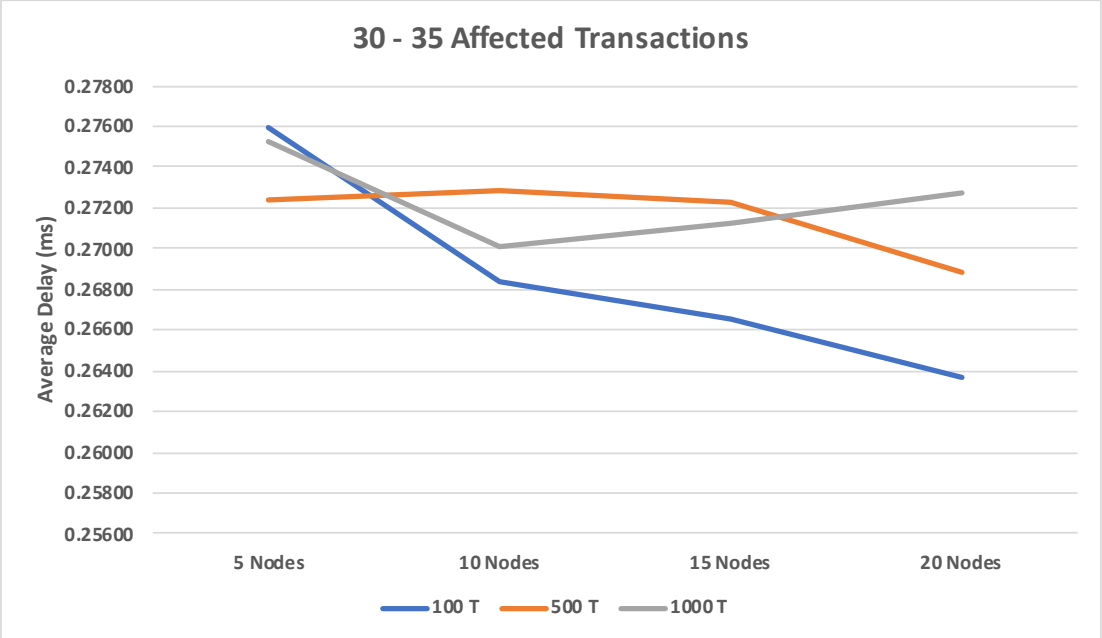

Figure 5.17: The impact of a different number of transactions and fog nodes on a set of 30 -35 affected Transactions

5.7.1.4 Fourth Experiment: Resource Requirement Cost

Table 5.4 shows the space required to store the global graph file on the trusted fog node and the local graph files on each pr_fog. 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 pr_fog 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.

Table 5.4: Storage requirement in Bytes for graph files with existing of trusted fog node

Number Of Fog Nodes	Storage Requirement In Bytes					
	100 Transactions		500 Transactions		1000 Transactions	
	Global Graph	Total Local Log Files	Global Graph	Total Local Log Files	Global Graph	Total Local Log Files
5	3,857	29,535	38,423	177,345	90,978	370,110
10	7,351	65,200	79,425	400,750	182,027	842,220
15	12,403	96,600	121,354	603,750	283,796	1302,885
20	16,725	125,800	166,822	806,580	386,372	1,738,640

5.7.2 Evaluation of the Third Model: Intelligent Government County System That Use Public Fog Nodes Distribution Model

5.7.2.1 Fifth Experiment: The impact of the different sets of affected transactions on various fog nodes number

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 3.6.3.1, 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 5.6.

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 5.7.1 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. The first group, fewer than five affected transactions were contained. The second group comprised of about ten to fifteen transactions that were affected, while the third grouping contained about thirty

to thirty-five 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.

Figs.5.18, 5.19, 5.20, and 5.21 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 milliseconds 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 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 held by the log file.

5.7.2.2 Sixth Experiment: The impact of different number of transactions on various number of fog nodes

Figures 5.22, 5.23, 5.24, and 5.25 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 does not have a significant impact on the total delay time which affects system availability. But, still, there is some small impact.

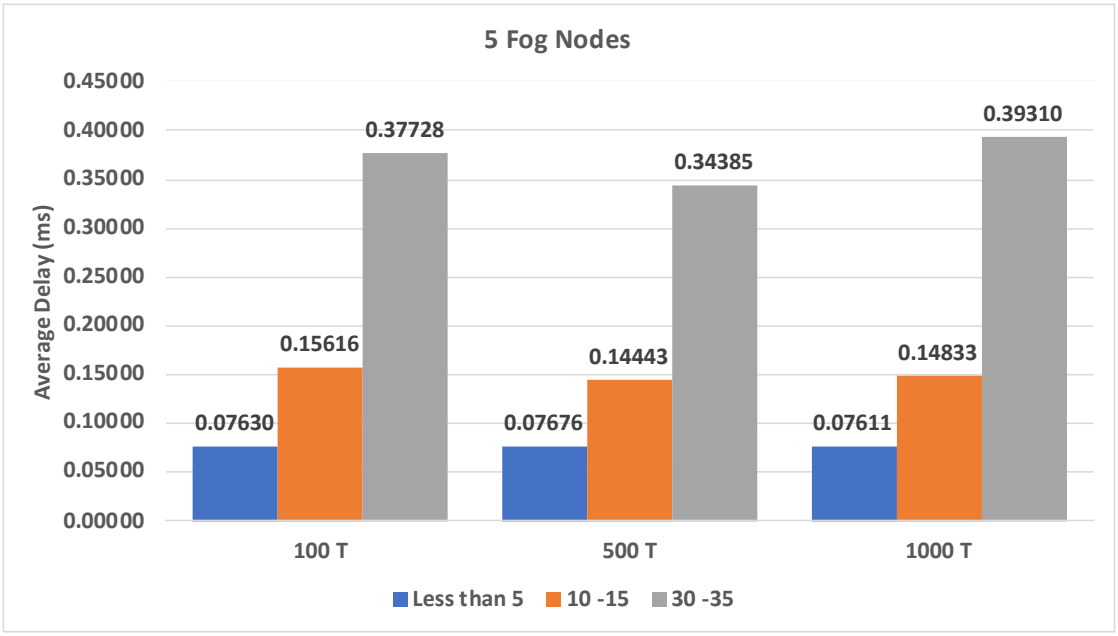

Figure 5.18: The impact of the different sets of affected transactions on five fog nodes

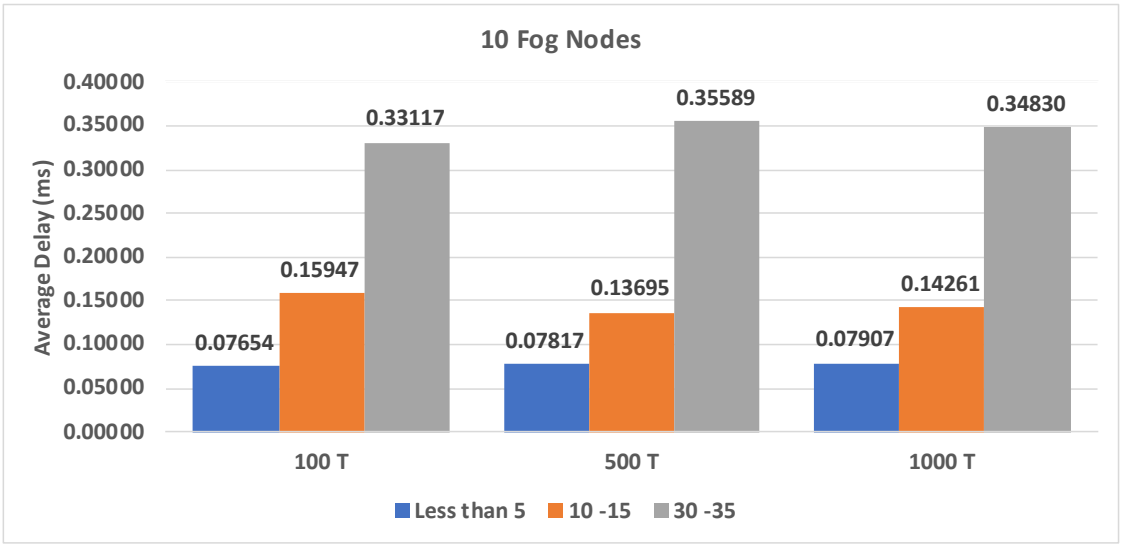

Figure 5.19: The impact of the different sets of affected transactions on ten fog nodes

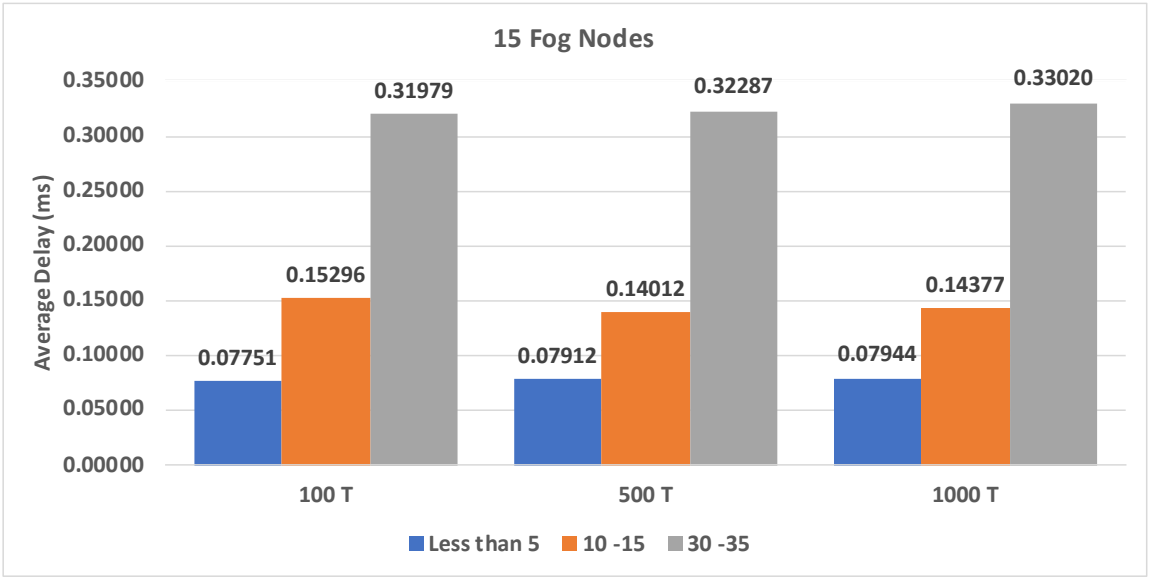

Figure 5.20: The impact of the different sets of affected transactions on fifteen fog nodes

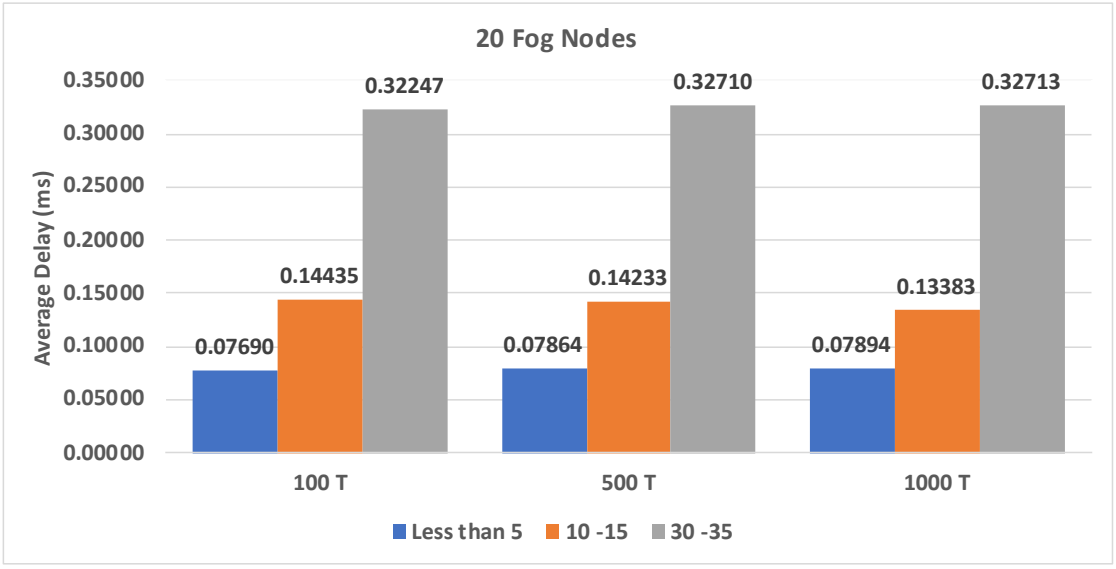

Figure 5.21: The impact of the different set of affected transactions on twenty fog nodes

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.

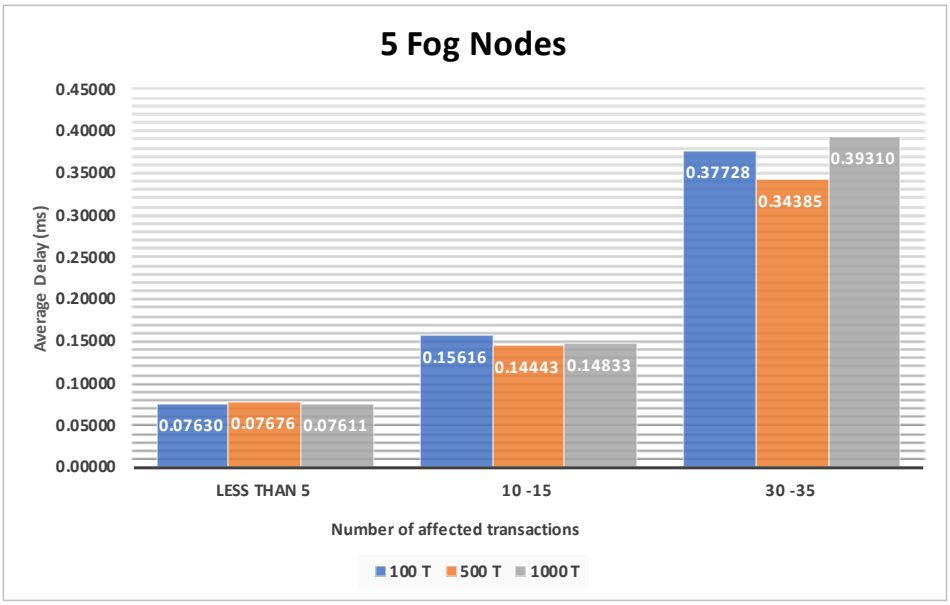

Figure 5.22: The impact of different number of transactions on five fog nodes

5.7.2.3 Seventh Experiment: The impact of a different number of transactions and fog nodes on different sets of affected transactions

Figs. 5.26, 5.27 and 5.28 illustrate the impact of a variable number of transactions on log file and a variable number of fog nodes. The sets of affected transactions used in this

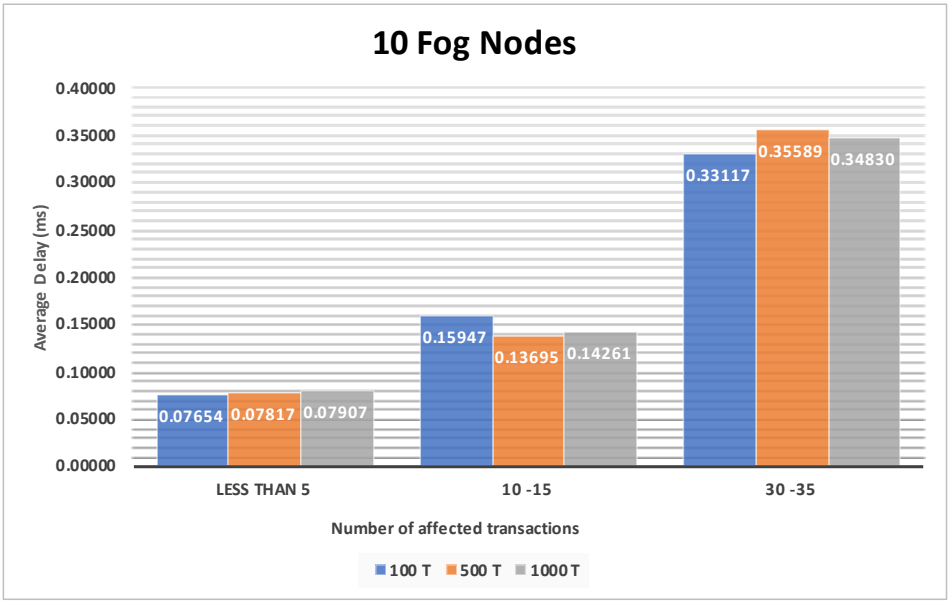

Figure 5.23: The impact of different number of transactions on ten fog nodes

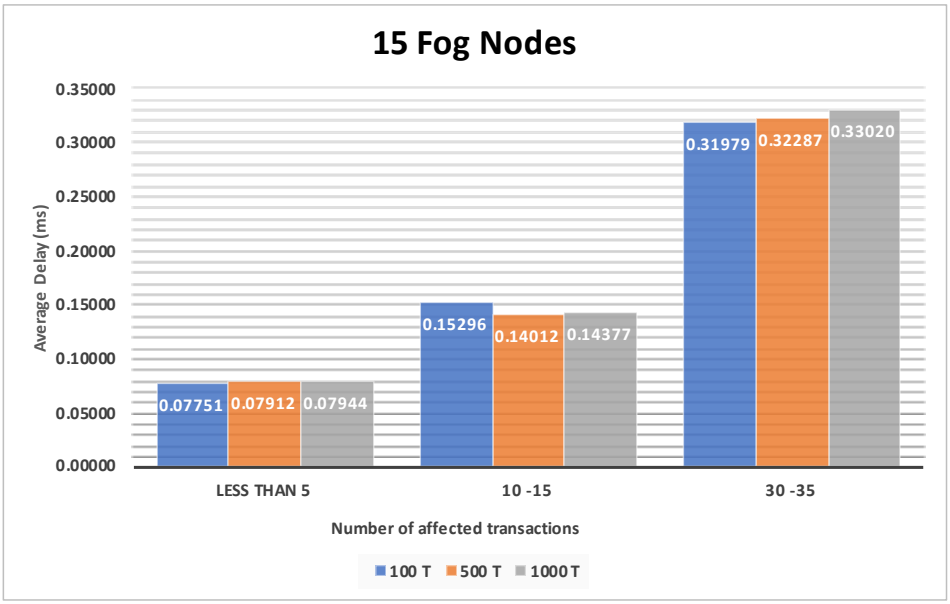

Figure 5.24: The impact of different number of transactions on fifteen fog nodes

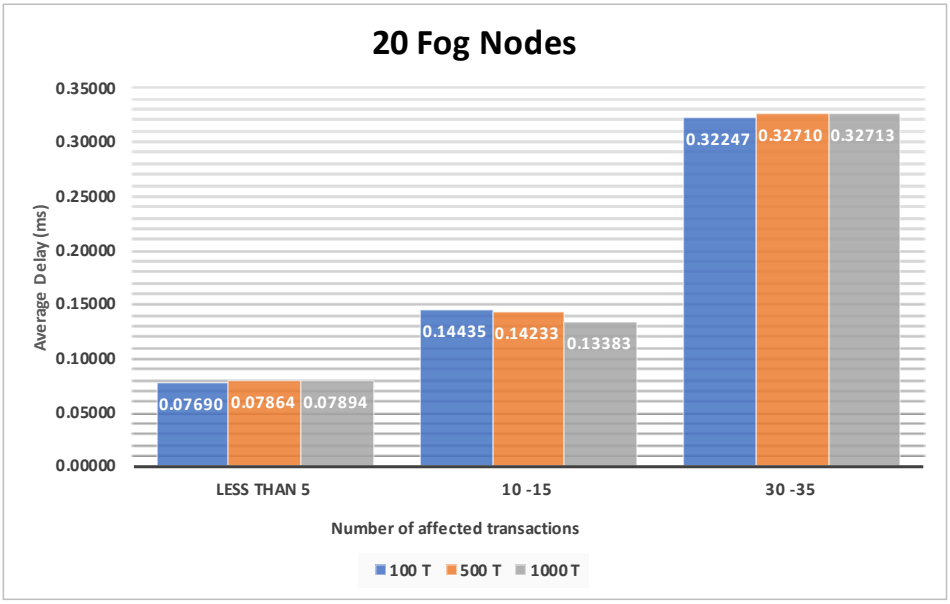

Figure 5.25: The impact of different number of transactions on twenty fog nodes

experiment were fixed as previously established.

Fig. 5.26 indicates the set of affected transactions as less than five. The time needed for our damage assessment algorithm to detect all affected transactions increased when the number of fog nodes increased. This scenario held true regardless of log file size.

The delay that occurred here can be attributed to two primary determinants. First, a small log file will impact fewer fog nodes than a large log file. Secondly, the local graphs expand with a rise in transaction dependency. We observed the local graphs in this model increasing in size with both the number of transactions on each log file and the number of fog nodes. We also observed that the number of transactions in each log file had little impact on the results. Hence, the larger log files of 1000 and 500 transactions took only slightly more time than those of 100 transactions when we used the set of less than five affected transactions.

However, the results differed, as seen in Fig. 5.27, when the affected transactions set was fixed at 10 to 15. Here we see in log files of 100 and 1000 transactions that an increase in the number of fog nodes decreases the delay time. Also, the log file of 100 transactions involves more time than the larger log files of 1000 and 500 transactions. But the time is slightly decreased when the number of fog nodes increases. Generally, what we found is that the larger set of affected transactions extends on the local graph levels with a small log file and fewer fog nodes. This explains the greater delay found in this configuration than in that of the graphs with large log files and more fog nodes.

Thus, the log files of 1000 and 500 transactions perform better than the log file of 100 transactions for two reasons. First, because all affected transactions in the system with the largest of the log files could be found in perhaps one or two levels of the affected graph, but with the smaller log file the damage could be found in more than two levels of the affected graphs. Second, with the smaller log file the damage will affect more fog nodes and that means more graphs are scanned.

Additionally, in this set of affected transactions, the log file of 500 transactions performs much better than the log file of 1000 transactions because the graph size of 1000 transactions is much larger than the graph size of 500 transactions and necessitates more time to locate a malicious transaction on the graph.

The set of 30 to 35 affected transactions produced different findings as seen in Fig. 5.28. The log file of 500 transactions exhibited the same behavior in processing this set as it did in the previous set of 10 to 15 affected transactions. Where the number of fog nodes is five, this log file showed better performance compared to the other two log files. Then, the delay time decreased a bit for the log files of 100 and 1000 transactions when the number of

fog nodes increased to 10. All the log files at 15 fog nodes closed the gap, and then remained constant and close at 20 fog nodes. As explained previously, the growth on the graph size, the increase in the level of affected transactions on the graphs, and the number of fog nodes that have been affected, have almost the same impact on the delay time.

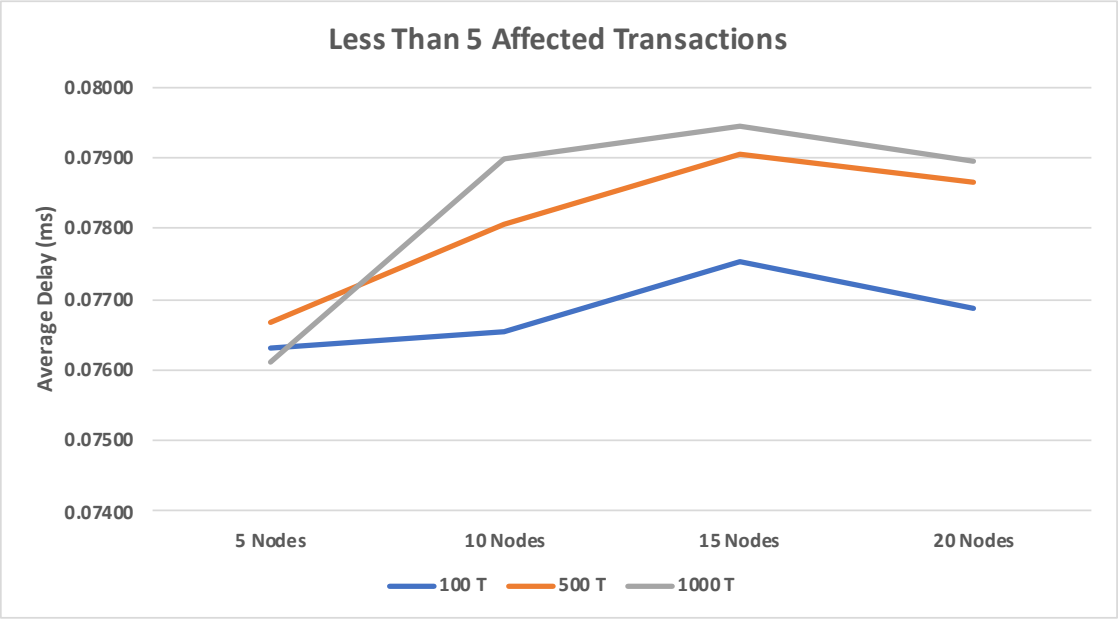

Figure 5.26: The impact of a different number of transactions and fog nodes on a set of less than five affected transactions

5.7.2.4 Eighth Experiment: Cost of Resource Requirements

The space needed for the storage of local graph files on every fog node is shown in Table 5.5. A growth in the size of the local graph files is a result of an increase in the number of transactions in the fog nodes or rather an augmentation of the number of fog nodes. However, our experiment's largest graph file was about 15847 kilobytes, which would be considered as a small size and would, therefore not result in any problems related to space. This makes the solution derived from the investigation inexpensive because of

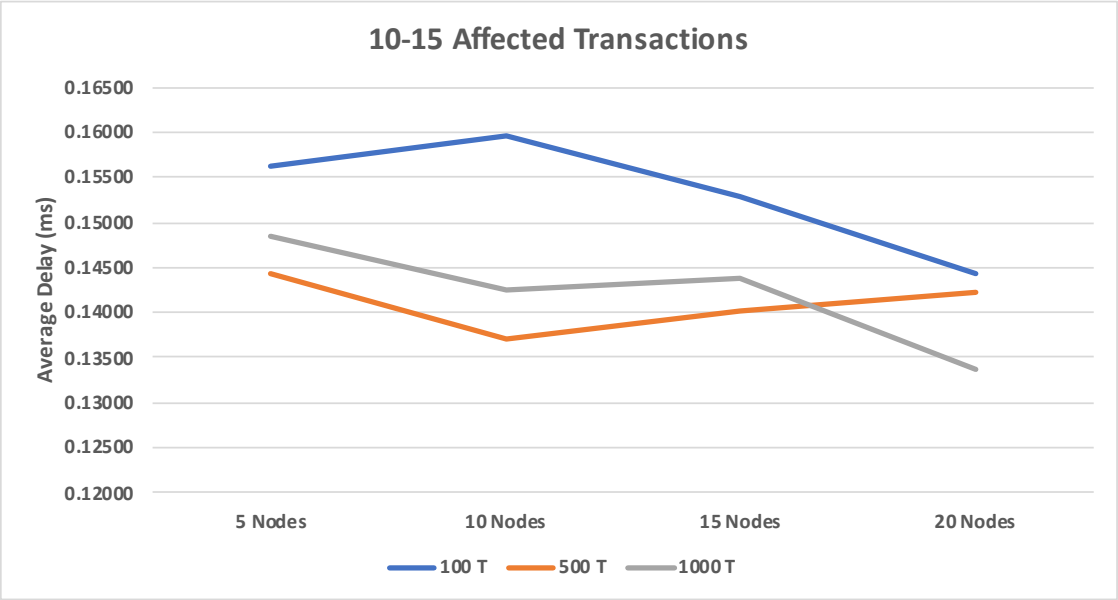

Figure 5.27: The impact of different number of transactions and fog nodes on set of ten to fifteen affected Transactions

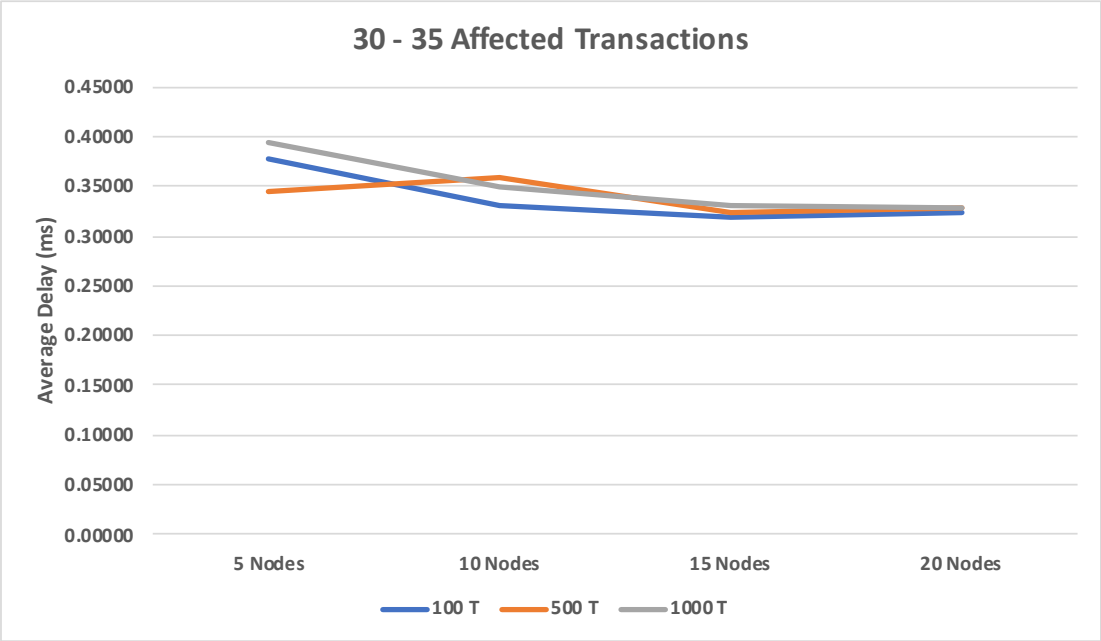

Figure 5.28: The impact of a different number of transactions and fog nodes on a set of 30 -35 affected Transactions

its space, because it would not require massive storage needs, making it appropriate for application.

Table 5.5: Storage requirement in Bytes for local graph files with absence of trusted fog node

Number Of Fog Nodes	Storage Requirement In Bytes		
	100 Transactions	500 Transactions	1000 Transactions
5	677	6,594	15,254
10	683	6,822	15,392
15	681	6,852	15,491
20	695	7104	15,847

5.7.3 Ninth Experiment: Comparison between having trusted fog node and not:

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 is no trusted fog node in the system (no global graph) Which is the better performing of the two, and under which factors? Figs. 5.29, 5.30, and 5.31 show the overall comparison between these models on the average runtime. The result, illustrated by Fig. 5.29, indicates that the second model, where there is no trusted fog node, performs 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. Also the damage caused by the small set of less than five affected transactions will usually not affect many fog nodes. On average only one to two fog nodes will be affected. The second model stands out a bit from the first model in this case.

However, as seen in Figs. 5.30, and 5.31, 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. And it is faster than the model that has 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 is required to scan the graph of each affected fog node where the first model needs to only scan the global graph.

In conclusion, the system benefits from having a trusted fog node when an attack compromises the database and damage spreads to more than five transactions. Further, the presence of the global graphs in the system stabilizes the results and speeds the detection process, minimizing the system's unavailability.

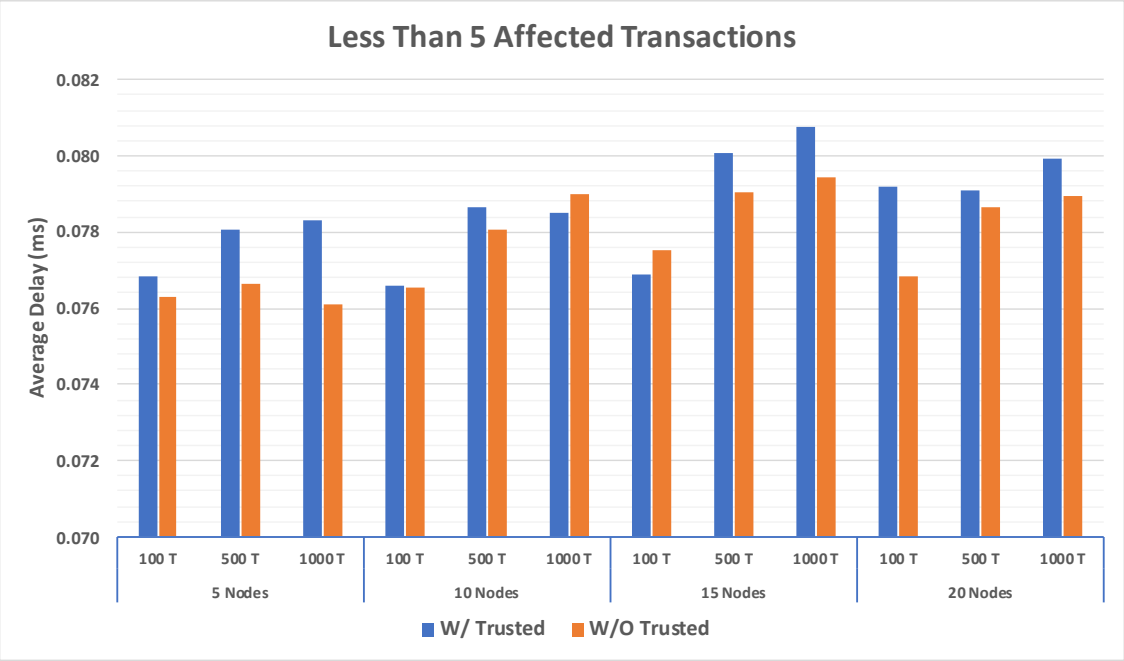

Figure 5.29: Comparison between having global graph on trusted fog node and not on a set of less than five affected transactions

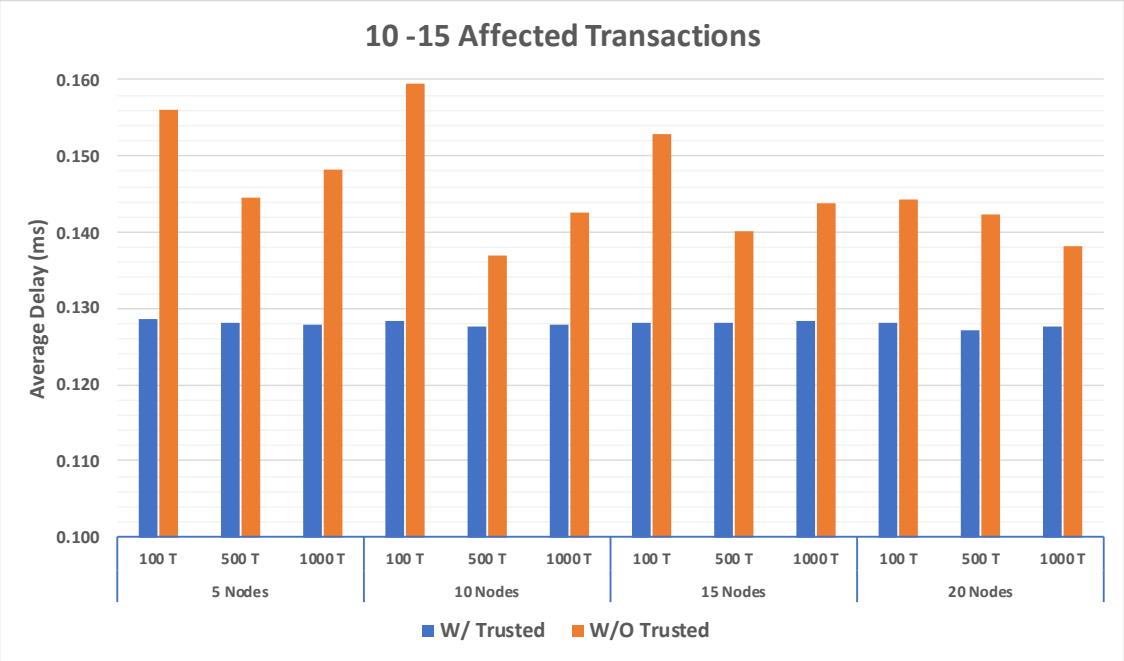

Figure 5.30: Comparison between having global graph on trusted fog node and not on set of ten to fifteen affected Transactions

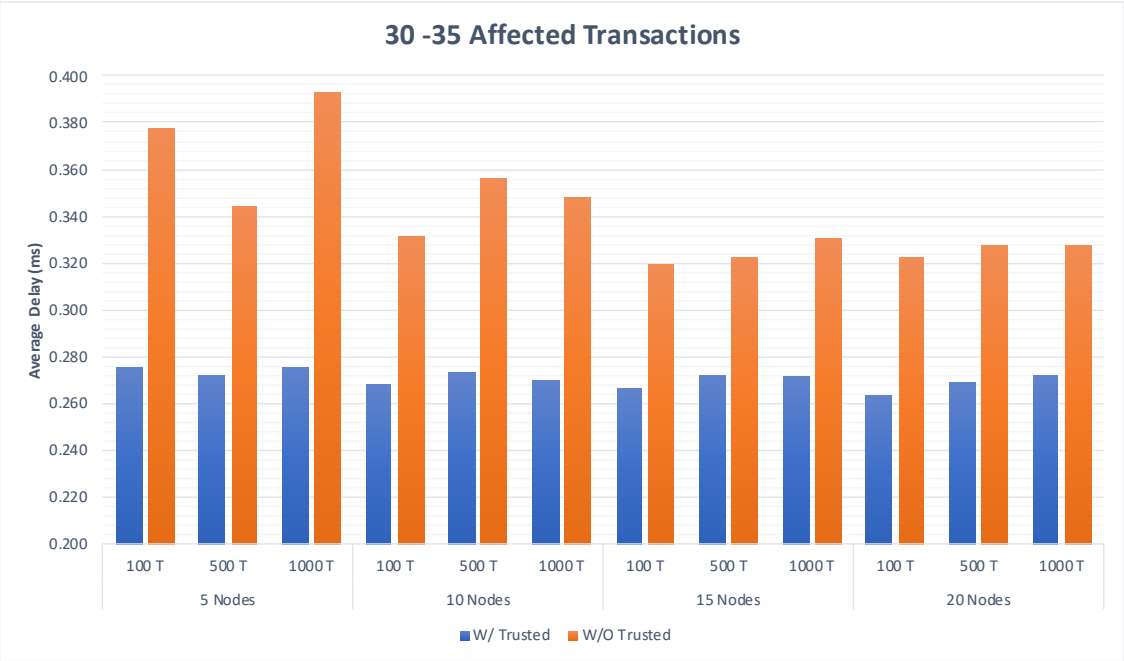

Figure 5.31: Comparison between having global graph on trusted fog node and not on a set of 30 -35 affected Transactions

6 Conclusion

Fog computing provides benefits to computing performance that traditional cloud systems do not. Data management in modern smart systems is well-positioned to significantly benefit from the utilization of fog computing. However, as any other data-sharing system, fog computing is vulnerable to attack and the injection of malicious transactions into the database. Intrusion detection is one of the main phases that must be included to ensure the security and reliability of any computing system. This phase uses software or device to observe the system for any malicious activity or policy violation. However, detection systems sometimes fail to detect several malicious transactions on time, leading to data damage. Therefore, intrusion detection must be complemented by another phase, namely, damage assessment and data recovery, which ensures the integrity and availability of system data. This phase identifies any further affected transactions and ensures that the database returns to a consistent state. Once the intrusion detection detects the malicious activities, an appropriate mechanism to assess and recover the damaged data from that attack is required and should be applied at the earliest opportunity.

Data damage assessment and recovery are fundamental to creating secure and reliable databases. This is particularly true for critical and sensitive data environments such as healthcare systems, critical infrastructure systems, and intelligent government systems. When an attack on these vital databases violates the integrity of stored data, the incident will result in serious consequences, including damage to property and even loss of life. Therefore, in addition to developing appropriate mechanisms for adaptation of fog computing in

intelligent systems, meeting security requirements to respond to attacks by providing fast and accurate damage assessment and recovery techniques is definitely necessary. The focus of this research has been the design of novel models for applying fog technology to modern smart systems. Working with the nature and characteristics of each model, we propose a unique approach sustaining the integrity of system data in the event of a cyberattack. Those approaches are designed to maintain the security of systems attacked by malicious transactions or subjected to fog node data modifications.

For the healthcare systems model, two different sub-models for applying fog technology to healthcare systems were introduced: fog modules with heterogeneous data, and fog modules with homogeneous data Working with the nature and characteristics of each model, we propose unique approach of assessing damaged data. Then we simulated and evaluated these two sub-models to demonstrate the difference between them and to discover, in consideration of all variables, which of the two is the better performer. We conclude that the model with homogeneous data stabilizes the results and speeds the detection process, minimizing the system's unavailability. Although, the model with heterogeneous data is needed as well because it is more appropriately applicable to most situations.

For critical infrastructure smart systems model, mainly emphasized the design of a unique technique for a smart system that uses fog computing technology to control and manage data. It also proposed a approach that use data dependency to gain the required information about the damaged part of the affected fog nodes database in order to assess damage to data caused by an attack. Thus, original data can be recovered, and a database can be returned to its consistent state as no attacking has occurred to ensure the integrity of consumer data in a fog computing environment in critical infrastructure systems. The model

and the proposed approach were designed and implemented to prove its applicability and to fulfill our goals by running several experiments. First part of experiments aimed to analyze the behavior of the proposed damage assessment algorithm as it detected affected data items with varying factors designed for the experiment. And the second half of the experiments is to measures the performance of our proposed data recovery algorithms in regaining all damaged data items and restoring their correct values as if no attack occurred.

For the last model we focus on designing novel model for an intelligent government system that uses fog computing technology to control and manage data. Unique algorithms that use transaction-dependency graph is implemented in this model to observe and monitor the activities of every transaction. Once an intrusion detection system detects malicious activities, the system will promptly detect all affected transactions. Two sub-models were introduced one requires trusted fog node and the other not. The one with trusted fog node can be applicable only on the private trusted fog node distribution where the second one is appropriate for the public or mixed fog nodes distribution. When a malicious transaction is found, the system is able to identify all affected transactions quickly. Several experiments were also performed to show the difference between those two mechanisms: one with a trusted fog node (global graph) present in the system and the other one which no trusted fog node is required in the system. Finally, we have shown the overall comparison between them on the average runtime, and which one of them is the better performing, and under which factors.

Bibliography

[1] Statista Research Department. (2016) Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions). [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] A. Holst. (2019) Data volume of internet of things (iot) connections worldwide in 2018 and 2025 (in zettabytes). [Online]. Available: https://www.statista.com/statistics/1017863/worldwide-iot-connected-devices-data-size/

[3] IHS Markit. (2017) The internet of things: a movement, not a market.

[4] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud," in *7th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[5] H.-N. Dai, H. Wang, G. Xu, J. Wan, and M. Imran, "Big data analytics for manufacturing internet of things: opportunities, challenges and enabling technologies," *Enterprise Information Systems*, vol. 0, no. 0, pp. 1–25, 2019. [Online]. Available: https://doi.org/10.1080/17517575.2019.1633689

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[7] G. Sun, S. Sun, J. Sun, H. Yu, X. Du, and M. Guizani, "Security and privacy preservation in fog-based crowd sensing on the internet of vehicles," *Journal of Network and Computer Applications*, vol. 134, pp. 89–99, 2019.

[8] CISCO. (2015) Fog computing and the internet of things: Extend the cloud to where the things are.

[9] ——. (2015) Cisco fog computing solutions- unleash the power of the internet of things.

[10] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, 2019.

[11] O. Akrivopoulos, I. Chatzigiannakis, C. Tselios, and A. Antoniou, "On the deployment of healthcare applications over fog computing infrastructure," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2. IEEE, 2017, pp. 288–293.

[12] D. G. Kirch and K. Petelle, "Addressing the physician shortage: the peril of ignoring demography," *Jama*, vol. 317, no. 19, pp. 1947–1948, 2017.

[13] Y. Cao, S. Chen, P. Hou, and D. Brown, "Fast: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation," in *2015 IEEE international conference on networking, architecture and storage (NAS)*. IEEE, 2015, pp. 2–11.

[14] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcare–a review and discussion," *IEEE Access*, vol. 5, pp. 9206–9222, 2017.

[15] U.S. Department of Health Human Services, Public Health Law. (last reviewed: September 14, 2018) Health insurance portability and accountability act of 1996 (hipaa).

[16] A. Paul, H. Pinjari, W.-H. Hong, H. C. Seo, and S. Rho, "Fog computing-based iot for health monitoring system," *Journal of Sensors*, vol. 2018, 2018.

[17] M. Aazam, P. P. Hung, and E.-N. Huh, "Smart gateway based communication for cloud of things," in *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*. IEEE, 2014, pp. 1–6.

[18] I. Azimi, A. Anzanpour, A. Rahmani, T. Pahikkala, M. Levorato, P. Liljeberg, and N. Dutt, "Hich: Hierarchical fog-assisted computing architecture for healthcare iot," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 174, 2017.

[19] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[20] S. Al-Janabi, I. Al-Shourbaji, M. Shojafar, and S. Shamshirband, "Survey of main challenges (security and privacy) in wireless body area networks for healthcare applications," *Egyptian Informatics Journal*, vol. 18, no. 2, pp. 113–122, 2017.

[21] Sandra Khvoynitskaya. (06/04/2020) Fog computing: shaping the future of smart cities.

[22] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Computing Surveys (CSUR)*, vol. 50, no. 3, p. 32, 2017.

[23] M. Marjani, F. Nasaruddin, A. Gani, A. Karim, I. A. T. Hashem, A. Siddiqa, and I. Yaqoob, "Big iot data analytics: architecture, opportunities, and open research challenges," *IEEE Access*, vol. 5, pp. 5247–5261, 2017.

[24] F. Y. Okay and S. Özdemir, "A fog computing based smart grid model," *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, 2016.

[25] M. Frustaci, P. Pace, G. Aloi, and G. Fortino, "Evaluating critical security issues of the iot world: Present and future challenges," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2483–2495, Aug 2018.

[26] J. LaPiedra, "The information security process prevention," *Detection and Response, Global Information Assurance Certification Paper, GIAC directory of certified professionals*, 2011.

[27] R. Elmasri and S. Navathe, *Fundamentals of database systems*. Pearson, 2017.

[28] F. Bonomi, "Connected vehicles, the internet of things, and fog computing," in *The eighth ACM international workshop on vehicular inter-networking (VANET), Las Vegas, USA*, 2011, pp. 13–15.

[29] H. Madsen, B. Burtschy, G. Albeanu, and F. Popentiu-Vladicescu, "Reliability in the utility computing era: Towards reliable fog computing," in *2013 20th International Conference on Systems, Signals and Image Processing*. IEEE, 2013, pp. 43–46.

[30] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcare–a review and discussion," *IEEE Access*, vol. 5, pp. 9206–9222, 2017.

[31] R. Mahmud, R. Kotagiri, and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*. Springer, 2018, pp. 103–130.

[32] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 416–464, Firstquarter 2018.

[33] T. Kudo, "Fog computing with distributed database," in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2018, pp. 623–630.

[34] Y. Kim, D. Kim, J. Son, W. Wang, and Y. Noh, "A new fog-cloud storage framework with transparency and auditability," in *2018 IEEE International Conference on Communications (ICC)*. IEEE, 2018, pp. 1–7.

[35] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 5, pp. 27–32, 2014.

[36] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *2014 Federated Conference on Computer Science and Information Systems*. IEEE, 2014, pp. 1–8.

[37] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury, and V. Kumar, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, pp. 19 293–19 304, 2017.

[38] F. Y. Okay and S. Ozdemir, "A secure data aggregation protocol for fog computing based smart grids," in *2018 IEEE 12th International Conference on Compatibility, Power Electronics and Power Engineering (CPE-POWERENG 2018)*. IEEE, 2018, pp. 1–6.

[39] S. Khan, S. Parkinson, and Y. Qin, "Fog computing security: a review of current applications and security solutions," *Journal of Cloud Computing*, vol. 6, no. 1, p. 19, 2017.

[40] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125–1142, 2017.

[41] D. Wu and N. Ansari, "A cooperative computing strategy for blockchain-secured fog computing," *IEEE Internet of Things Journal*, 2020.

[42] D. Wu and N. Ansari, "A cooperative computing strategy for blockchain-secured fog computing," *IEEE Internet of Things Journal*, pp. 1–1, 2020.

[43] A. Alazeb and B. Panda, "Maintaining data integrity in fog computing based critical infrastructure systems," in *Proceedings of the 2019 International Conference on Computational Science and Computational Intelligence*, ser. CSCI'19. Las Vegas, Nevada, USA: IEEE Computer Society, 2019, p. 40–47.

[44] L. Zhu, M. Li, Z. Zhang, C. Xu, R. Zhang, X. Du, and N. Guizani, "Privacy-preserving authentication and data aggregation for fog-based smart grid," *IEEE Communications Magazine*, vol. 57, no. 6, pp. 80–85, 2019.

[45] L. Lyu, K. Nandakumar, B. Rubinstein, J. Jin, J. Bedo, and M. Palaniswami, "Ppfa: privacy preserving fog-enabled aggregation in smart grid," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3733–3744, 2018.

[46] M. Aazam, S. Zeadally, and K. A. Harras, "Deploying fog computing in industrial internet of things and industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4674–4682, 2018.

[47] R. Lu, K. Heung, A. H. Lashkari, and A. A. Ghorbani, "A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced iot," *IEEE Access*, vol. 5, pp. 3302–3312, 2017.

[48] A. Alazeb and B. Panda, "Ensuring data integrity in fog computing based health-care systems," in *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 2019, pp. 63–77.

[49] O. Consortium *et al.*, "Openfog reference architecture for fog computing," *Architecture Working Group*, pp. 1–162, 2017.

[50] M. Alshehri and B. Panda, "An encryption-based approach to protect fog federations from rogue nodes," in *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 2019, pp. 225–243.

[51] ——, "A blockchain-encryption-based approach to protect fog federations from rogue nodes," in *2019 3rd Cyber Security in Networking Conference (CSNet)*. IEEE, 2019, pp. 6–13.

[52] J. Vora, S. Tanwar, S. Tyagi, N. Kumar, and J. J. P. C. Rodrigues, "Faal: Fog computing-based patient monitoring system for ambient assisted living," in *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*, 2017, pp. 1–6.

[53] V. Vijayakumar, D. Malathi, V. Subramaniyaswamy, P. Saravanan, and R. Logesh, "Fog computing-based intelligent healthcare system for the detection and prevention of mosquito-borne diseases," *Computers in Human Behavior*, vol. 100, pp. 275–285, 2019.

[54] P. G. V. Naranjo, Z. Pooranian, M. Shojafar, M. Conti, and R. Buyya, "Focan: A fog-supported smart city network architecture for management of applications in the internet of everything environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 274–283, 2019.

[55] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, and Q. Yang, "A hierarchical distributed fog computing architecture for big data analysis in smart cities," in *Proceedings of the ASE BigData & SocialInformatics 2015*, 2015, pp. 1–6.

[56] D. Amaxilatis, I. Chatzigiannakis, C. Tselios, N. Tsironis, N. Niakas, and S. Papadogeorgos, "A smart water metering deployment based on the fog computing paradigm," *Applied Sciences*, vol. 10, no. 6, p. 1965, 2020.

[57] I. Froiz-Míguez, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, "Design, implementation and practical evaluation of an iot home automation system for fog computing applications based on mqtt and zigbee-wifi sensor nodes," *Sensors*, vol. 18, no. 8, p. 2660, 2018.

[58] P. Wang, S. Liu, F. Ye, and X. Chen, "A fog-based architecture and programming model for iot applications in the smart grid," *arXiv preprint arXiv:1804.01239*, 2018.

[59] G. Pernul, "Database security," in *Advances in Computers*. Elsevier, 1994, vol. 38, pp. 1–72.

[60] B. Panda and K. A. Haque, "Extended data dependency approach: a robust way of rebuilding database," in *Proceedings of the 2002 ACM symposium on Applied computing*. ACM, 2002, pp. 446–452.

[61] G. Fu, H. Zhu, and Y. Li, "A robust damage assessment model for corrupted database systems," in *International Conference on Information Systems Security*. Springer, 2009, pp. 237–251.

[62] U. P. Rao and D. R. Patel, "Incorporation of application specific information for recovery in database from malicious transactions," *Information Security Journal: A Global Perspective*, vol. 22, no. 1, pp. 35–45, 2013.

[63] M. Xie, H. Zhu, Y. Feng, and G. Hu, "Tracking and repairing damaged databases using before image table," in *2008 Japan-China Joint Workshop on Frontier of Computer Science and Technology*. IEEE, 2008, pp. 36–41.

[64] B. Panda and A. Alazeb, "Securing database integrity in intelligent government systems that employ fog computing technology," in *2020 International Conference on Computing and Data Science (CDS)*. IEEE, 2020, pp. 202–207.

[65] S. Kaddoura, R. A. Haraty, A. Zekri, and M. Masud, "Tracking and repairing damaged healthcare databases using the matrix," *Int. J. Distrib. Sen. Netw.*, vol. 2015, Jan. 2016. [Online]. Available: https://doi.org/10.1155/2015/914305

[66] P. Ammann, S. Jajodia, and P. Liu, "Recovery from malicious transactions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 5, pp. 1167–1185, 2002.

[67] P. Liu and M. Yu, "Damage assessment and repair in attack resilient distributed database systems," *Computer Standards & Interfaces*, vol. 33, no. 1, pp. 96–107, 2011.

[68] A. Chakraborty, A. K. Majumdar, and S. Sural, "A column dependency-based approach for static and dynamic recovery of databases from malicious transactions," *International Journal of Information Security*, vol. 9, no. 1, pp. 51–67, 2010.

[69] B. Panda and J. Giordano, "Reconstructing the database after electronic attacks," in *Database Security XII*. Springer, 1999, pp. 143–156.

[70] Y. Zuo and B. Panda, "Distributed database damage assessment paradigm," *Information management & computer security*, vol. 14, no. 2, pp. 116–139, 2006.

[71] R. Sobhan and B. Panda, "Reorganization of the database log for information warfare data recovery," in *Database and Application Security XV*. Springer, 2002, pp. 121–134.

[72] C. Lala and B. Panda, "Evaluating damage from cyber attacks: a model and analysis," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 31, no. 4, pp. 300–310, 2001.

[73] B. Panda and Jing Zhou, "Database damage assessment using a matrix based approach: an intrusion response system," in *Seventh International Database Engineering and Applications Symposium, 2003. Proceedings.*, 2003, pp. 336–341.

[74] K. Kurra, B. Panda, W.-N. Li, and Y. Hu, "An agent based approach to perform damage assessment and recovery efficiently after a cyber attack to ensure e-government database security," in *2015 48th Hawaii International Conference on System Sciences*. IEEE, 2015, pp. 2272–2279.

[75] B. Panda and R. Yalamanchili, "Transaction fusion in the wake of information warfare," in *Proceedings of the 2001 ACM symposium on Applied computing*, 2001, pp. 242–247.

[76] X. Xia, Q. Ji, and J. Le, "Research on transaction dependency mechanism of self-healing database system," in *2012 International Conference on Systems and Informatics (ICSAI2012)*. IEEE, 2012, pp. 2357–2360.

[77] J. Zheng, X. Qin, and J. Sun, "Data dependency based recovery approaches in survival database systems," in *International Conference on Computational Science*. Springer, 2007, pp. 1131–1138.

[78] Y. Son, H. Kang, H. Han, and H. Y. Yeom, "Improving performance of cloud key-value storage using flushing optimization," in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE, 2016, pp. 42–47.

[79] S. Kontopoulos and G. Drakopoulos, "A space efficient scheme for persistent graph representation," in *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. IEEE, 2014, pp. 299–303.

[80] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[81] A. Alazeb, B. Panda, S. Almakdi, and M. Alshehri, "Data integrity preservation schemes in smart healthcare systems that use fog computing distribution," *Electronics*, vol. 10, no. 11, p. 1314, 2021.

[82] Z. Khan, Z. Pervez, and A. Ghafoor, "Towards cloud based smart cities data security and privacy management," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. IEEE, 2014, pp. 806–811.

[83] C. Chang, S. Narayana Srirama, and R. Buyya, "Indie fog: An efficient fog-computing infrastructure for the internet of things," *Computer*, vol. 50, no. 9, pp. 92–98, 2017.

7 Publications

Publications Reprint Permissions:

Reprinted by permission from [Abdulwahab Alazeb and Brajendra Panda]: [Springer] [Security, Privacy, and Anonymity in Computation, Communication, and Storage (SpaCCS 2019)] [Ensuring Data Integrity in Fog Computing Based Health-Care Systems, Abdulwahab Alazeb and Brajendra Panda] [© Springer, Cham] 2019

©[2019] IEEE. Reprinted, with permission, from [Abdulwahab Alazeb and Brajendra Panda, Maintaining data integrity in fog computing based critical infrastructure systems, 12/2019]

©[2020] IEEE. Reprinted, with permission, from [Abdulwahab Alazeb and Brajendra Panda, Securing Database Integrity in Intelligent Government Systems that Employ Fog Computing Technology, 08 /2020].

1) "Ensuring data integrity in fog computing based health-care systems." Alazeb, Abdulwahab, and Brajendra Panda. International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage. https://doi.org/10.1007/978-3-030-24907-6_6. Springer, 2019, Atlanta, USA. Published.

2) "Maintaining data integrity in fog computing based critical infrastructure systems." Alazeb, Abdulwahab, and Brajendra Panda. 2019 International Conference on Computational Science and Computational Intelligence (CSCI), doi: 10.1109/CSCI49370.2019.00014. IEEE, 2019, Las Vegas, USA. Published.

3) "Securing Database Integrity in Intelligent Government Systems that Employ Fog Computing Technology." Panda, Brajendra, and Abdulwahab Alazeb. 2020 International Conference on Computing and Data Science (CDS), doi: 10.1109/CDS49703.2020.00048. IEEE, 2020, Stanford, CA, USA. Published.

4) "Data Integrity Preservation Schemes in Smart Healthcare Systems That Use Fog Computing Distribution." Alazeb, A.; Panda, B.; Almakdi, S.; Alshehri, M. Electronics 2021, 10, 1314. https://doi.org/10.3390/electronics10111314. Published