University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Mitigating Poisoning Attacks in Decentralized Federated Learning through Moving Target Defense

Zi Ye
Zurich, Switzerland
Student ID: 21-737-119

Supervisor: Chao Feng, Dr. Alberto Huertas Celdran
Date of Submission: March 25, 2024

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, _____

Signature of student

Abstract

Decentralized Federated Learning (DFL) allows collaborative model training while preserving data privacy, but it is vulnerable to poisoning attacks. This thesis proposes Dynamic Aggregation Functions, a Moving Target Defense (MTD) strategy, to improve DFL's resilience against such attacks. The main contributions are: 1) Providing a comprehensive theoretical background on Federated Learning, poisoning attacks, and MTD; 2) Developing a framework for understanding MTD techniques in DFL, exploring implementation perspectives like shuffling, diversity, redundancy and hybrid techniques; 3) Proposing the novel Dynamic Aggregation mechanism that enables nodes to proactively or reactively switch aggregation functions. Evaluations on benchmark datasets demonstrate the effectiveness of proactive MTD in low poisoning scenarios, with limitations in high poisoning environments. The techniques have negligible overhead on system resources.

This thesis is one of the first to investigate MTD for addressing security issues in DFL. The proposed Dynamic Aggregation method enhances DFL's resilience to poisoning attacks. The thesis also acknowledges the limitations and suggests future research directions, such as evaluating the performance in non-IID data scenarios and integrating Dynamic Aggregation with other security mechanisms.

Zusammenfassung

Dezentrales föderiertes Lernen (DFL) ermöglicht kollaboratives Modelltraining unter Wahrung der Datenprivatsphäre, ist jedoch anfällig für Vergiftungsangriffe (Poisoning Attacks). Diese Masterarbeit schlägt dynamische Aggregationsfunktionen vor, eine Moving Target Defense (MTD) Strategie, um die Widerstandsfähigkeit von DFL gegen solche Angriffe zu verbessern. Die Hauptbeiträge sind: 1) Bereitstellung eines umfassenden theoretischen Hintergrunds zu föderiertem Lernen, Vergiftungsangriffen und MTD; 2) Entwicklung eines Frameworks zum Verständnis von MTD-Techniken in DFL, das Implementierungsperspektiven wie Shuffling, Diversity, Redundanz und Hybrid-Techniken untersucht; 3) Vorschlag des neuartigen Mechanismus der dynamischen Aggregation, der es Knoten ermöglicht, Aggregationsfunktionen proaktiv oder reaktiv zu wechseln. Auswertungen auf Benchmark-Datensätzen zeigen die Wirksamkeit von proaktivem MTD in Szenarien mit geringer Vergiftung, mit Einschränkungen in Umgebungen mit hoher Vergiftung. Die Techniken haben einen vernachlässigbaren Overhead auf Systemressourcen.

Diese Arbeit ist eine der ersten, die MTD zur Lösung von Sicherheitsproblemen in DFL untersuchen. Die vorgeschlagene Methode der dynamischen Aggregation verbessert die Widerstandsfähigkeit von DFL gegen Vergiftungsangriffe. Die Arbeit erkennt auch die Grenzen an und schlägt zukünftige Forschungsrichtungen vor, wie die Bewertung der Leistung in nicht-IID-Datenszenarien und die Integration der dynamischen Aggregation mit anderen Sicherheitsmechanismen.

Acknowledgments

First, I would like to extend my sincere gratitude to my supervisors Chao Feng and Dr. Alberto Hueartas for their expert guidance and thoughtful feedback throughout our bi-weekly meetings. Further, I am also grateful to Prof. Dr. Stiller for allowing me to write my thesis at the Communication Systems Group. Last, special thanks to Enrique Tomás Martinez Beltrán, for his open-source platform, Fedstellar.

Contents

Chapter 1

Introduction

Federated Learning (FL) is an emerging paradigm in Machine Learning (ML) that enables collaborative training of models across distributed datasets while preserving data privacy[1]. In FL, instead of sharing raw data, participants share locally trained model updates, which are then aggregated to obtain a global model. FL can be categorized into two types: Centralized Federated Learning (CFL)[1] and Decentralized Federated Learning (DFL)[2]. While CFL relies on a central server for model aggregation, DFL eliminates this single point of failure by allowing decentralized aggregation among neighboring participants[3].

Despite its advantages, DFL is vulnerable to adversarial attacks, particularly poisoning attacks[4][5]. In poisoning attacks, malicious participants intentionally inject biased or harmful data to compromise the integrity of the global model[2][6]. These attacks can prevent model convergence or cause convergence to a flawed model, leading to misclassifications or backdoor vulnerabilities[5].

1.1 Motivation

Federated Learning (FL) is a novel approach in Machine Learning (ML) that enables collaborative learning on distributed datasets while preserving user privacy[7]. FL can be categorized into two types: Centralized Federated Learning (CFL) and Decentralized Federated Learning (DFL)[1][2]. Although CFL is widely used in various domains, it suffers from drawbacks such as a single point of failure and bottleneck. DFL was introduced in 2018 to address these issues by allowing decentralized aggregation of model parameters, eliminating the need for a central server[8]. However, DFL is vulnerable to adversarial attacks, where malicious clients can manipulate local data or the model to compromise the accuracy and robustness of the global model[6].

One of the significant threats to DFL is poisoning attacks, where an attacker manipulates the training data or model updates to degrade the performance of the global model[6]. To tackle this challenge, this thesis proposes the integration of Moving Target Defense (MTD) techniques to enhance the security and resilience of DFL systems against poisoning

attacks. MTD is a security concept that involves constantly changing the attack surfaces to impede the ability of adversaries to execute successful attacks[5].

1.2 Description of Work

The Master Thesis aims to design and implement a Moving Target Defense (MTD)-based defense mechanism to improve the robustness and security of the Decentralized Federated Learning (DFL) system. The project is divided into four main phases:

Background Research and Problem Understanding: In this phase, this work will acquire the conceptual elements involved in the proposal of the system and understand the background knowledge of DFL, its challenges, vulnerabilities, and the MTD mechanism. This includes studying the fundamentals of DFL, identifying the vulnerabilities and challenges associated with DFL, and understanding the MTD mechanism and its potential to enhance the security of DFL systems.

Design and Architecture of the MTD-based Defense Mechanism: In this phase, this work will present a proposal for the design and architecture of the MTD-based defense mechanism for the DFL. This includes identifying the features and configurations that must be incorporated into the proposed system to enhance the resilience of the DFL framework. The student must consider factors such as the ability to run on resource-constrained containers or devices, compatibility with existing DFL frameworks (e.g., FedStellar), and the ability to support MTD mechanisms (e.g., dynamic topology, dynamic communication port, or dynamic aggregation function). The proposed approaches must be thoroughly discussed with the supervisors to determine the most optimal strategy and make informed design decisions that align with the objectives of the project.

Prototyping and Implementation: In this phase, this work will construct and execute the MTD-based defense module utilizing the predetermined features and configurations to safeguard the DFL system against poisoning attacks. The student must take into account the practicability of implementing such an MTD mechanism in devices with limited resources. The resulting system must be thoroughly documented in the design and implementation sections of the report, including the exclusion of alternative approaches and the rationale behind their exclusion.

Evaluation and Conclusion: In this phase, this work will evaluate and deliberate upon the proposed system and its implementation from multiple perspectives, utilizing a range of metrics. These metrics will serve to illustrate the efficacy of the novel approach, specifically in terms of employing the MTD mechanism to mitigate a targeted set of poisoning attacks. These evaluations must be conducted in coordination with the supervisors during regular meetings. A final report must include a motivation and problem description, background information, related work, design decision, implementation details, evaluation, and conclusions.

In summary, the goal of this Master Thesis is to design and implement a Moving Target Defense (MTD)-based defense mechanism to improve the robustness and security of the

Decentralized Federated Learning (DFL) system. The thesis aims to address the vulnerability of DFL to adversarial attacks, where malicious clients can manipulate local data or the model to compromise the accuracy and robustness of the global model. The proposed MTD-based defense mechanism will be designed to enhance the resilience of the DFL framework by constantly changing the attack surfaces to perplex and deceive adversaries, thereby impeding their ability to execute successful attacks.

1.3 Thesis Outline

The thesis is organized into seven chapters:

Chapter 1: Introduction This chapter provides an overview of the motivation behind the research, a description of the work to be done, and an outline of the thesis. The motivation section discusses the importance of securing decentralized federated learning (DFL) systems against poisoning attacks and the potential of moving target defense (MTD) mechanisms to address this challenge. The description of work section outlines the research objectives, methodology, and expected outcomes. The thesis outline section provides a brief overview of the remaining chapters.

Chapter 2: Background This chapter presents the background knowledge of DFL, challenges and vulnerabilities of DFL, and the MTD mechanism. The DFL section discusses the concept of DFL, its advantages and disadvantages, and its applications. The challenges and vulnerabilities section discusses the security risks associated with DFL, including poisoning attacks. The MTD mechanism section discusses the concept of MTD, its benefits, and its applications in cybersecurity.

Chapter 3: Related Work This chapter reviews the existing literature on MTD-based defense mechanisms for machine learning (ML), centralized federated learning (CFL), and DFL. The ML section discusses the use of MTD in securing ML models against adversarial attacks. The CFL section discusses the use of MTD in securing CFL systems against poisoning attacks. The DFL section discusses the existing research on MTD-based defense mechanisms for DFL systems.

Chapter 4: Defense Design This chapter presents the design and architecture of the proposed MTD-based defense mechanism for DFL. The design section discusses the design considerations and trade-offs, while the architecture section presents the overall structure of the proposed defense mechanism. The chapter also discusses the selection of MTD techniques and their integration into the DFL system.

Chapter 5: Implementation This chapter describes the implementation of the proposed defense mechanism. The implementation section discusses the technical details of the implementation, including the programming language, libraries, and tools used. The chapter also presents the results of the implementation, including the performance and scalability of the proposed defense mechanism.

Chapter 6: Evaluation This chapter evaluates the proposed system and its implementation from multiple perspectives. The evaluation section discusses the evaluation method-

ology, metrics, and results. The chapter also discusses the limitations of the proposed defense mechanism and potential avenues for future research.

Chapter 7: Summary and Conclusions This chapter summarizes the findings of the thesis and provides conclusions. The summary section provides a brief overview of the research objectives, methodology, and outcomes. The conclusions section discusses the implications of the research findings and their potential impact on the field of DFL security. The chapter also discusses the limitations of the research and potential avenues for future work.

Chapter 2

Background

Machine learning (ML) is a rapidly advancing technology that is being widely adopted for various applications. Unlike traditional model training, which requires storing all user data in centralized servers, federated learning (FL) has become popular due to its ability to store data in a distributed manner. By sharing the model parameters instead of data, Federated Learning (FL) is gaining more popularity in the Artificial Intelligene (AI) community. FL can be centralized, known as Centralized Federated Learning (CFL), or decentralized, known as Decentralized Federated Learning (DFL). DFL allows for decentralized aggregation of model parameters by neighboring participants, eliminating the need for a central server. However, DFL systems are vulnerable to adversarial attacks, particularly poisoning attacks, where malicious participants intentionally inject biased or harmful data to compromise the integrity of the global model. Moving Target Defense (MTD) is a security concept that involves constantly changing the attack surfaces to impede adversaries' ability to execute successful attacks.

The following sections will delve into the theoretical foundations of FL, its framework structure, and its two primary branches: Centralized Federated Learning (CFL) and Decentralized Federated Learning (DFL). However, FL also introduces new attack surfaces. The challenges and risks that FL encounters in terms of security, as well as the potential attack methods that adversaries may employ will be introduced next. Afterwards, an introduction to several defense mechanisms aimed at enhancing the security of FL will be provided, with particular emphasis on Moving Target Defense (MTD).

2.1 Federated Learning

The widespread adoption of communication technologies and the growing prevalence of the Internet of Things (IoT) concept have led to a substantial increase in the number of devices connected to the internet. Most people often carry smart personal devices equipped with various sensors (such as cameras, microphones, accelerometers, and GPS chips). Therefore, personal computing devices provide an opportunity to access a large amount of training data, which is crucial for building reliable machine learning models. Traditional machine learning requires collecting training data on a single machine or

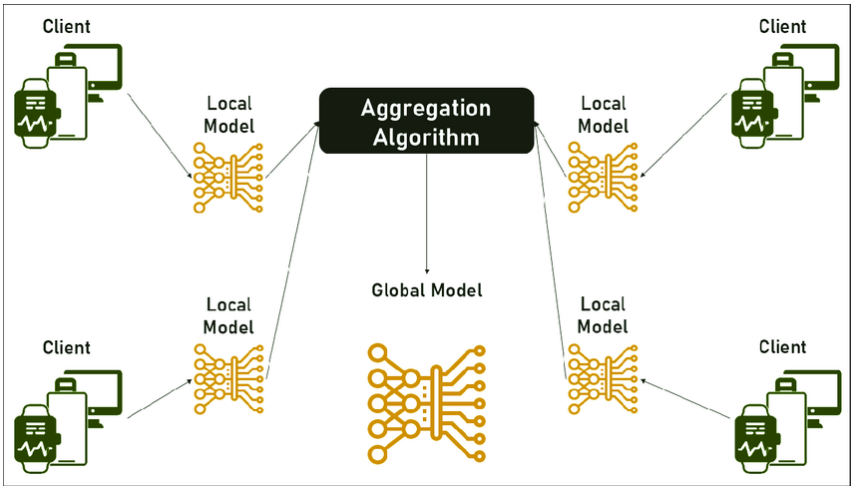


Figure 2.1: The main process of federal learning [13]

data center. Therefore, technology companies must go through an expensive and time-consuming process to obtain their users' data, not to mention the risks and responsibilities associated with storing data in a centralized location [9], posing new challenges in processing and analyzing such data in a privacy-preserving manner. Federated Learning (FL) is a technique that allows entities to train collaborative models without sharing sensitive data [10]. Unlike traditional Machine Learning (ML) methods, which require gathering the training data in a single machine or in a data center, FL allows users to share their local models instead of the raw data with a central server, preserving data privacy [11]. Federated Learning can help solve data privacy, communication latency, and scalability issues, as sensitive data does not need to leave its original location [12]. Figure 2.1 depicts a conceptual model for federated learning, illustrates how the clients, local models, global model, and aggregation algorithm interact in a decentralized way to enable collaborative model training while keeping the training data private and localized on each client device.[13]. Federated Learning is particularly applicable in scenarios with high privacy requirements, such as healthcare and financial services, it can also be applied to internet of things(IoT), and natural language processing[3]. But Federated Learning also faces challenges. There are four fundamental challeges in FL: Expensive Communication, Systems Heterogeneity, Statistical Heterogeneity and Privacy Concerns [14].

2.1.1 Federated Learning Basics

Aggregation in Federated Learning (FL) is a crucial process that involves combining the learning updates (like model weights or gradients) from multiple distributed nodes (like devices or servers) to create a global model. This process is central to the federated learning paradigm, which allows for collaborative learning without sharing raw data among participants. The aggregation step typically occurs on a central server or aggregator, and there are various methods for aggregating model updates in Federated Learning. The choice of aggregation method can impact the performance, privacy, and convergence properties of the federated learning system.

But how is Federated Learning training actually performed? Stochastic Gradient Descent

(SGD) has shown excellent results in deep learning. Therefore, as a baseline, researchers decided to base the training algorithm of federated learning on SGD as well. SGD can be naively applied to federated optimization problems, that is, only one batch gradient calculation is performed in each round of communication (for example, on randomly selected clients). This approach is computationally efficient but requires a large number of training iterations to produce high-quality models. "FedAvg" or Federated Averaging is an algorithm which combines local stochastic gradient descent (SGD) on each client with a server that performs model averaging [10]. It has established itself as the de facto standard algorithm for FL. Figure 2.2 gives a pseudo-code of Federated Averaging Algorithm. To understand it, a few terms need to be defined first.

- w_t - Model weights on communication round t
- w_t^k - Model weights on communication round t on client k
- C - Fraction of clients performing computations in each round
- E - Number of training passes each client makes over its local dataset on each round
- B - The local minibatch size used for the client updates
- η - The learning rate
- P_k - Set of data points on client k
- n_k - Number of data points on client k
- $f_i(w)$ - Loss $l(x_i, y_i; w)$ i.e., loss on example (x_i, y_i) with model parameters w

The baseline algorithm, FedSGD, short for Federated SGD, sets the parameter C to 1, which corresponds to a full-batch (non-stochastic) gradient descent. For the current global model w_t, the average gradient on its global model is calculated for each client k.

$$F_k(w) = \frac{1}{n_k} \sum_{i \in P_k} f_i(\omega) \tag{2.1}$$

$$g_k = \nabla F_k(w_t) \tag{2.2}$$

The central server then aggregates these gradients and applies the update.

$$w_{t+1} \Leftarrow w_t - \eta \sum_{k=1}^{k} \frac{n_k}{n} g_k \tag{2.3}$$

In the context of FedAvg, a modification is applied to the update step. Each client does not only send model parameters, but also runs several local gradient descent steps on the current model. The server then takes a weighted average of the updated models. This

Algorithm 1 `FederatedAveraging`. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
initialize w_0
for each round $t = 1, 2, \dots$ **do**
$m \leftarrow \max(C \cdot K, 1)$
$S_t \leftarrow$ (random set of m clients)
for each client $k \in S_t$ **in parallel do**
$w_{t+1}^k \leftarrow \text{ClientUpdate}(k, w_t)$
$w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate(k, w): *// Run on client k*
$\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size B)
for each local epoch i from 1 to E **do**
for batch $b \in \mathcal{B}$ **do**
$w \leftarrow w - \eta \nabla \ell(w; b)$
return w to server

Figure 2.2: FedAvg algorithm pseudocode [10]

is the Federated Averaging (FedAvg) algorithm, which puts more computational work on individual clients.

$$\forall\ k, w_{t+1}^k \Leftarrow\ w_t - \eta g_k \tag{2.4}$$

$$w_{t+1}\ \Leftarrow\ \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k \tag{2.5}$$

The reason for making this change is to achieve major speedups in practice by improving the computation on each client once a minimum level of parallelism over clients is reached. The amount of computation is controlled by three parameters: C (the fraction of clients participating in that round), E (the number of training passes each client makes over its local dataset each round), and B (the local minibatch size used for client updates)

2.1.2 Centralized Federated Learning (CFL)

Federated learning (FL) enables collaborative training of machine learning models without sharing individual data, addressing privacy concerns in data-rich environments. However, different FL architectures vary in terms of communication overhead, scalability, and privacy guarantees.

The most common FL architecture is centralized federated learning (CFL), where a central server coordinates the training process and aggregates the model updates from the clients[15][3]. In CFL, the server initializes a global model and sends it to the participating clients. Each client then trains the model on their local data and sends the updated model parameters back to the server. The server aggregates the updates, typically by averaging, to improve the global model[15][16]. This process is repeated for multiple rounds until the model converges or a desired performance is reached. CFL algorithms, such as Federated Averaging (FedAvg)[10], have demonstrated effective performance in various applications. However, the centralized architecture has some limitations, such as communication bottlenecks, single point of failure, and potential privacy risks at the server level[17][18].

2.1.3 Decentralized Federated Learning (DFL)

To address the limitations of CFL, decentralized federated learning (DFL) has emerged as an alternative approach[17][19]. In DFL, the clients communicate directly with each other in a peer-to-peer manner, without relying on a central server for aggregation. The clients are connected in a communication network, often represented as a graph, and exchange model updates with their neighbors[17][20]. Each client aggregates the received updates locally and updates its own model accordingly. DFL eliminates the need for a central coordinator, reducing communication costs and improving robustness against node failures[19][21]. However, DFL introduces new challenges, such as designing efficient communication protocols, ensuring convergence, and handling data and system heterogeneity[22].

2.1.4 Comparison between CFL and DFL

While CFL and DFL differ in their architectures and workflows, they share the common goal of enabling collaborative training while preserving data privacy. To better understand their trade-offs, let's compare the two approaches across several key aspects Image 2.3 clearly illustrates the key differences between Centralized Federated Learning (CFL) and Decentralized or Peer-to-Peer Federated Learning (DFL).

In part (a), the diagram shows a centralized model where data is sent to a central server for training. The process involves the following steps:

1. Local training with local data.
2. Send model updates to the server.
3. Aggregate model updates.
4. Send updated model back to clients.

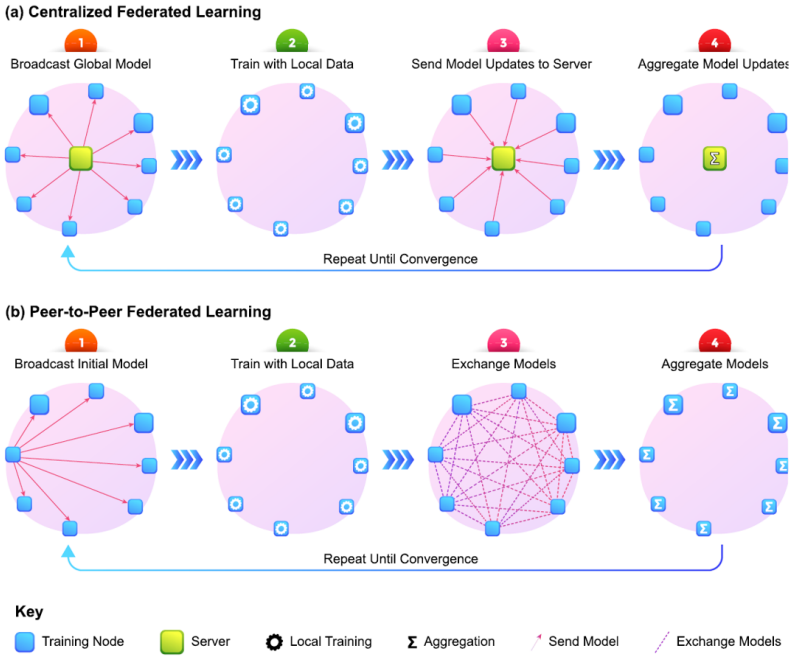


Figure 2.3: FL Workflow [23]

The arrows indicate the flow of data and models between the clients and the server.

In part (b), the diagram illustrates a peer-to-peer model where data is exchanged directly between clients without a central server. The process includes:

1. Local training with local data.
2. Exchange models with other clients.
3. Repeat until convergence.

The arrows in this diagram show the direct exchange of models between clients.

In summary, part (a) depicts the traditional centralized federated learning where a server orchestrates the training, while part (b) shows the decentralized peer-to-peer federated learning where nodes collaborate directly without a central coordinator.

To further compare and contrast these two approaches, this work also examines the key aspects summarized in the following table: 2.1. This table compares the key aspects of Centralized Federated Learning (CFL) and Decentralized Federated Learning (DFL):

In summary, DFL provides advantages in terms of robustness, communication efficiency, privacy, and scalability, while CFL has benefits in terms of convergence speed, simpler implementation, and better theoretical understanding. The choice between the two depends on the specific requirements and constraints of the application scenario.

Aspect	CFL	DFL
Architecture	Uses a central server to coordinate training and aggregate model updates from clients	No central server, clients communicate directly with each other to exchange model updates
Robustness	Single point of failure at the central server	More robust, avoids bottlenecks and failures of a central server
Communication Cost	Higher, all updates go through the central server	Lower, updates exchanged directly between clients
Privacy	Central server has access to all client updates, potentially weaker privacy	Stronger privacy, each client only communicates with a subset of other clients
Scalability	Limited by the capacity of the central server	More scalable, supports larger number of clients
Convergence Speed	Generally faster due to global aggregation at the server	Can be slower, depends on the peer-to-peer communication topology
Client Participation	Suitable for a large number of clients, e.g. cross-device FL	Supports flexible and ad-hoc participation of clients
Implementation Complexity	Simpler to implement and manage due to central coordination	More complex, requires carefully designed peer-to-peer protocols
Heterogeneity Handling	Central server can enforce policies to handle data and system heterogeneity	More challenging, needs to account for variability across clients
Theoretical Analysis	Better understood convergence properties	Active area of research, providing convergence guarantees is harder

Table 2.1: Comparison between Centralized Federated Learning (CFL) and Decentralized Federated Learning (DFL)

2.2 Poisoning Attacks

Although FL is considered effective in protecting the privacy of users during training, further research shows that FL is also exposed to many security and privacy risks [11][23], such as poisoning attacks [24][25][26] and privacy leakage [27][28][29].

Poisoning attacks can be generally classified based on the adversary's goal and technique. Based on attack goals, there can be three types of attacks: untargeted, targeted, and backdoor attacks; Based on attack technique, poisoning attacks can be classified into data poisoning attacks and model poisoning attacks.[30][6][31]. Figure 2.4 provides a taxonomy of different types of poisoning attacks and highlights various techniques under each category, providing a comprehensive overview of the different ways machine learning systems can be compromised through poisoning. The following sections will explain each

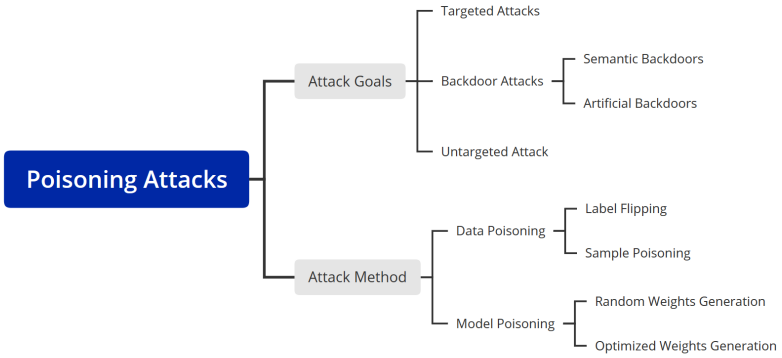


Figure 2.4: Taxonomy for poisoning attacks

types of attacks in detail.

2.2.1 Classification by Attack Goal

Poisoning attacks can be classified based on the attacker's goals, which include targeted attacks, untargeted attacks, and backdoor attacks.

Targeted Attacks Targeted attacks aim to manipulate the model's behavior for specific inputs without degrading its overall performance [30][32]. The attacker's goal is to cause misclassification or targeted misclassification of particular instances while maintaining the model's accuracy on other inputs. For example, an attacker might attempt to cause a facial recognition system to misclassify a specific individual while functioning normally for others.

Untargeted Attacks Untargeted attacks, also known as Byzantine attacks, aim to degrade the model's overall performance by adding noise or irrelevant data points to the training set [33]. The attacker's goal is to reduce the model's accuracy across various inputs without targeting specific instances. Untargeted attacks can be more difficult to detect as they do not exhibit a clear pattern of misclassification.

Backdoor Attacks Backdoor attacks involve inserting hidden triggers or patterns into the model during training, causing it to misbehave when the trigger is present while functioning normally otherwise [34], [35]. There are two main types of backdoor attacks:

- Semantic Backdoors: In semantic backdoor attacks, the attacker uses semantically meaningful triggers, such as specific objects or features, to activate the backdoor [36]. These attacks can be challenging to detect as the triggers blend in with the legitimate data distribution.
- Artificial Backdoors: Artificial backdoor attacks involve using artificially generated patterns or noise as triggers [37]. These triggers are not semantically meaningful but can still cause the model to misbehave when present. Artificial backdoors can be easier to detect compared to semantic backdoors due to their synthetic nature.

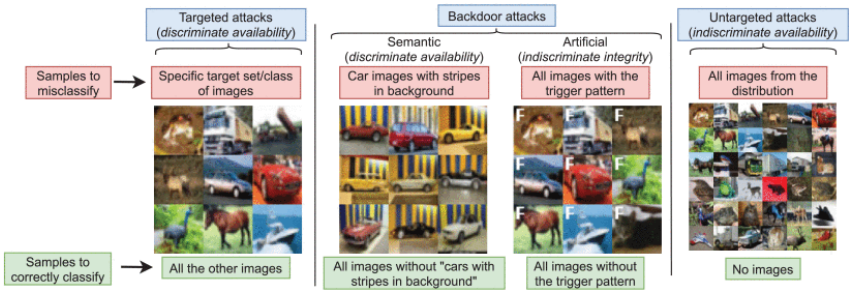


Figure 2.5: Classes of FL poisoning attacks and their objectives [38]

Figure 2.5 illustrates different types of poisoning attacks that can occur in machine learning systems, specifically focusing on image classification tasks. The image is divided into three main columns: Targeted Attacks, Backdoor Attacks, and Untargeted Attacks, effectively demonstrating the different objectives and characteristics of poisoning attacks,

2.2.2 Classification by Attack Technique

Poisoning attacks can also be classified based on the attack technique employed by the adversary, which includes data poisoning and model poisoning.

Data Poisoning Data poisoning attacks involve tampering with the training data by adding noise, flipping labels, or injecting malicious examples [39], [40]. Two common data poisoning techniques are:

- Label Flipping: In label flipping attacks, the attacker injects mislabeled data into the training set to bias the model's decision-making [41]. By strategically flipping labels, the attacker can manipulate the model's behavior towards a desired outcome.
- Sample Poisoning: Sample poisoning attacks involve modifying a significant portion of the training data with misleading or malicious examples [42]. The attacker aims to influence the model's learning process by introducing carefully crafted samples that steer the model towards a specific behavior.

Model Poisoning Model poisoning attacks target the model updates or parameters to arbitrarily deviate the model from its intended behavior [43][44]. Two common model poisoning techniques are:

- Random Weights Generation: In random weights generation attacks, malicious participants upload arbitrary model updates without contributing meaningful data or compute resources. These random updates can disrupt the model's convergence and degrade its performance.
- Optimized Weights Generation: Optimized weights generation attacks involve crafting model updates that are specifically designed to manipulate the model's behavior [35]. The attacker optimizes the weights to achieve a desired outcome, such as targeted misclassification or backdoor insertion.

2.2.3 Poisoning Attacks in Decentralized Federated Learning

Federated Learning (FL) has shown great potential for enabling collaborative learning among multiple parties who do not fully trust each other, as it allows them to train a shared global model without revealing their private data. However, FL systems are vulnerable to poisoning attacks, where adversarial participants can deliberately submit manipulated or corrupted model updates during the iterative training process [45]. These malicious contributions aim to sabotage the performance and utility of the final global model, for example, by degrading its accuracy or introducing targeted misclassifications. As a result, poisoning attacks pose a significant threat to the integrity and reliability of FL, undermining its practical applicability in real-world scenarios involving untrusted clients.

Decentralized Federated Learning (DFL) is particularly vulnerable to poisoning attacks due to its distributed nature and the involvement of multiple participants in the training process [10]. In DFL, participants contribute their local model updates to collaboratively train a global model without sharing raw data. This setting opens up new attack surfaces and challenges in detecting and mitigating poisoning attacks.

Malicious participants in DFL can exploit the lack of centralized control and the opacity of local training to inject poisoned updates into the global model [46]. These poisoned updates can propagate through the network, compromising the integrity and performance of the final model. Moreover, the decentralized nature of DFL makes it difficult to identify and isolate malicious participants, as there is no central authority to monitor and validate individual contributions.

Figure 2.6 illustrates the concept of poisoning attacks in federated learning (FL). It highlights the two main types of poisoning attacks in FL - data poisoning where the training data is corrupted, and model poisoning where the model updates are directly manipulated by the attacker. By injecting poisoned updates into the aggregation process at the server, the malicious user can degrade the performance of the global model or potentially insert backdoors or targeted misclassifications. This poses a significant security threat to federated learning systems.

Several studies have demonstrated the feasibility and impact of poisoning attacks in DFL. For example, Bhagoji et al. [43] showed that a single malicious participant can significantly degrade the accuracy of a federated learning model by injecting poisoned updates. Fang et al. [47] proposed a model poisoning attack called ”local model poisoning” that allows an attacker to manipulate the global model without direct access to other participants' data or models.

The dynamic nature of decentralized FL makes it vulnerable to poisoning attacks. Understanding the taxonomy and characteristics of poisoning attacks is crucial for developing effective defense mechanisms. The nest section will explore Moving Target Defense(MTD) as a potential mitigation approach against poisoning attacks in Decentralized Federated Learning, making it harder for attackers to succeed and providing a proactive defense strategy.

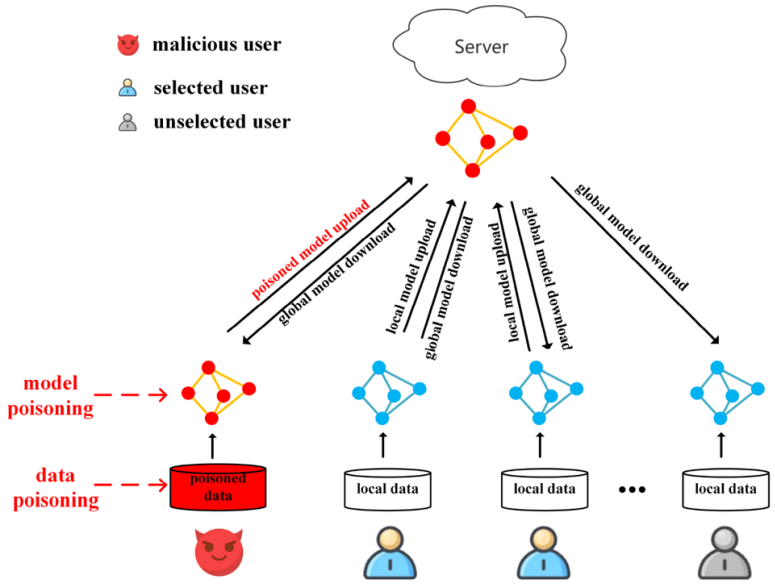


Figure 2.6: Poisoning Attacks in Federated Learning [30]

2.3 Moving Target Defense

2.3.1 Moving Target Defense Basics

Moving Target Defense (MTD) is an emerging proactive defense approach that aims to thwart attacks by continuously changing the attack surface[48]. The key idea behind MTD is to introduce controlled changes to the system configuration over time, thereby increasing the uncertainty and complexity for attackers[48]. By making the system less predictable and more dynamic, MTD reduces the window of opportunity for attackers and increases the cost of their probing and attack efforts[48][49].

The concept of MTD is inspired by the notion of "moving targets" in military tactics, where a constantly moving and changing target is more difficult to hit than a static one[48]. In the cyber domain, MTD techniques can be applied at different layers of the system stack, such as the network, platform, runtime environment, software, and data layers[48][50].

Figure 2.7 provides a taxonomy of various Moving Target Defense (MTD) techniques and strategies. This taxonomy provides a comprehensive overview of the various MTD techniques and strategies, highlighting the key components and approaches involved in creating a dynamic and unpredictable attack surface. By understanding and combining these techniques, organizations can develop effective MTD systems tailored to their specific security requirements and system architectures.

There are three elements for an MTD technique: WHAT to move, HOW to move, and WHEN to move.

WHAT to move refers to the system components that are vulnerable to exploitation, such as IP addresses, software, data, and system configurations[51]. By dynamically changing

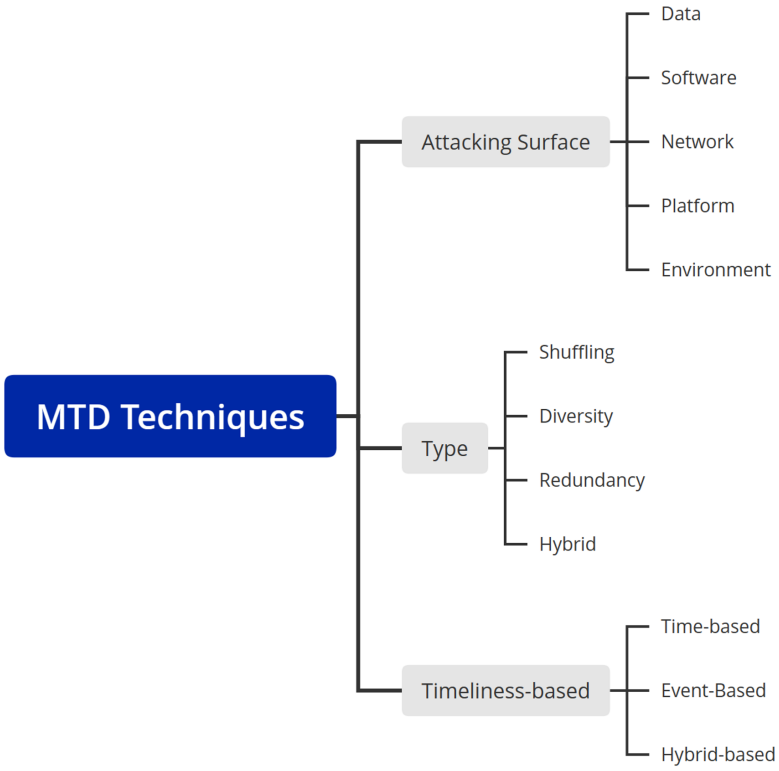


Figure 2.7: Taxonomy of Moving Target Defense Techniques

these components, MTD reduces the attack surface and makes it harder for attackers to target specific vulnerabilities[48], [51].

HOW to move involves techniques like randomization, diversity, and adaptation[48], [51]. Randomization introduces unpredictability, while diversity creates heterogeneity in the system components, making it more difficult for attackers to develop generic exploits[48]. Adaptation allows the system to respond to detected threats or changes in the environment[51].

WHEN to move concerns the strategic timing of changes, balancing security with operational impact[51]. MTD techniques can be triggered at fixed intervals, randomly, or in response to specific events or threat.

Key elements of MTD techniques[51][52]

1. WHAT to move: The configuration set (C) [51]
   (a) Data (encryption, representation, storage)
   (b) Software (versions, implementations)
   (c) Network configurations (IP addresses, ports, protocols)
   (d) Platform configurations (OS, hardware, VMs)
   (e) Runtime environments (memory layout, instruction sets)

2. HOW to move: The movement function (M) [53][48]
   (a) Shuffling
   (b) Diversity
   (c) Redundancy
   (d) Randomization
3. WHEN to move: The timing function (T) [51]
   (a) Time-based (fixed intervals, random intervals)
   (b) Event-triggered (security alerts, system changes)

2.3.2 Types of Moving Target Defense

MTD techniques can be applied to various aspects of a system, including data, software, network configurations, platform configurations, runtime environments, and hybrid approaches[48], [51], [52], [54].

Data Data-level MTD techniques focus on protecting the confidentiality, integrity, and availability of data through methods such as encryption, data representation, and storage[51], [52]. Encryption can be applied to data at rest and in transit, using dynamic encryption keys and algorithms to create a moving target for attackers[51]. Data representation techniques involve changing the format, encoding, or structure of data to make it harder for attackers to understand and manipulate[52]. Dynamic data storage techniques, such as data fragmentation and distributed storage, can be used to spread data across multiple locations, making it more difficult for attackers to compromise the entire dataset[51].

Software Software-level MTD techniques introduce diversity and unpredictability into the software components of a system[48], [54]. This can involve using different software versions, implementations, or configurations to create a heterogeneous environment that is more resistant to generic exploits[48]. Techniques such as N-version programming, where multiple functionally equivalent software versions are deployed simultaneously, can be used to create a moving target for attackers[54]. Additionally, dynamic software composition and adaptation techniques can be employed to automatically update and reconfigure software components in response to detected threats or changes in the environment[48].

Network configurations Network-level MTD techniques focus on dynamically changing network configurations, such as IP addresses, port numbers, and communication protocols[48], [51]. IP hopping involves periodically changing the IP addresses of network nodes, making it harder for attackers to identify and target specific systems[48]. Port hopping techniques dynamically change the port numbers used for communication, while protocol hopping switches between different communication protocols to create a moving target[51]. Software-Defined Networking (SDN) can be leveraged to create a more flexible and dynamic network infrastructure, enabling the implementation of various network-level MTD techniques[54].

Platform configurations Platform-level MTD techniques target the underlying infrastructure of a system, including the operating system, hardware, and virtual machines[51], [52]. Dynamic platform composition techniques involve creating a heterogeneous environment with multiple operating systems, hardware architectures, and virtualization technologies[52]. This diversity makes it harder for attackers to develop exploits that can compromise the entire system[51]. Additionally, techniques such as live migration of virtual machines and dynamic resource allocation can be used to create a moving target for attackers, making it harder to locate and exploit critical system components[51].

Runtime environments Runtime-level MTD techniques focus on creating unpredictability and diversity in the runtime environment of a system, such as memory layout and instruction sets[48], [54]. Address Space Layout Randomization (ASLR) is a commonly used technique that randomizes the memory locations of key system components, making it harder for attackers to exploit memory corruption vulnerabilities[48]. Instruction set randomization techniques involve dynamically changing the instruction set architecture or introducing new instructions to create a moving target for attackers[54]. These techniques can be combined with other runtime-level MTD approaches, such as dynamic code generation and execution, to further increase the unpredictability and diversity of the system[48].

Hybrid Hybrid MTD techniques combine multiple types of MTD approaches to create a more comprehensive and adaptable defense strategy[52], [54]. By leveraging the strengths of different MTD techniques across various system components, hybrid approaches can provide a more robust and effective defense against a wide range of threats[52]. For example, a hybrid MTD system might combine data-level techniques, such as encryption and data fragmentation, with network-level techniques, such as IP hopping and SDN-based reconfiguration, to protect both the data and the communication channels[54]. The choice of techniques in a hybrid MTD system depends on the specific security requirements, performance constraints, and system architecture of the protected environment[52].

2.3.3 Key Techniques in Moving Target Defense

MTD techniques can be categorized into several main approaches, including shuffling, diversity, redundancy, randomization, and hybrid techniques[48], [51], [52].

Shuffling techniques involve dynamically changing the mapping between system resources and their logical identities, such as IP addresses, port numbers, and memory locations[48], [52]. This creates a moving target for attackers, making it harder to identify and exploit specific resources[48].

Diversity techniques introduce heterogeneity into the system components, such as using different operating systems, software implementations, or hardware architectures[48], [51]. This reduces the effectiveness of generic exploits and forces attackers to develop multiple attack vectors, increasing the cost and complexity of attacks[48].

Redundancy techniques involve creating multiple instances of critical system components, such as servers, network paths, or data storage[52]. This allows the system to maintain availability and functionality even if some instances are compromised or disrupted[52].

Randomization techniques introduce unpredictability into the system, such as randomizing the memory layout, instruction sets, or communication channels[48], [51]. This makes it harder for attackers to rely on deterministic exploitation methods and increases the uncertainty in the attack process[48].

Hybrid techniques combine multiple MTD approaches to create a more comprehensive and adaptable defense strategy[52]. For example, a hybrid technique might use shuffling to dynamically change network configurations, diversity to introduce heterogeneity in software components, and redundancy to maintain system availability[52]. By leveraging the strengths of different techniques, hybrid approaches can provide a more robust and effective MTD solution[52].

The choice of MTD technique depends on various factors, such as the system architecture, performance requirements, and security objectives[51]. Each technique has its own advantages and challenges, and the most effective MTD strategy often involves a combination of techniques tailored to the specific needs of the system[48], [51], [52].

2.3.4 Timing Function of MTD Techniques

In addition to the various types of MTD techniques, the timing of when to apply these techniques is a critical consideration in designing an effective MTD system[48], [51]. The timing function of MTD techniques can be broadly categorized into time-based and event-triggered approaches[48], [52]. **Time-based** Time-based MTD techniques involve applying dynamic changes to the system at predetermined time intervals[48], [51]. These intervals can be fixed, where the changes occur at regular, static intervals, or random, where the intervals between changes vary according to a specified probability distribution[48]. Fixed-interval time-based MTD techniques are simple to implement and can provide a consistent level of protection over time[51]. However, they may be predictable, allowing attackers to synchronize their attacks with the known intervals between changes[52]. Random-interval time-based MTD techniques introduce an element of unpredictability, making it harder for attackers to anticipate and exploit the timing of changes[48]. However, the effectiveness of these techniques depends on the chosen probability distribution and the trade-off between security and system performance[51]. **Event-triggered** Event-triggered MTD techniques initiate dynamic changes in response to specific events or conditions within the system or its environment[48], [52]. These events can include security alerts, such as the detection of an attack or anomalous behavior, or system changes, such as the addition or removal of components[51]. Security alert-triggered MTD techniques allow the system to adapt its defense posture in real-time based on the detected threats[48]. For example, if an intrusion detection system (IDS) detects a potential attack, the MTD system can automatically reconfigure the network topology, update firewall rules, or deploy deceptive assets to mitigate the threat[52]. System change-triggered MTD techniques ensure that the system maintains a consistent level of protection as its components and configurations evolve[51]. For instance, when a new device is added to the network, the MTD system can automatically assign it a random IP address and configure its security settings to match the current defense posture[48]. Event-triggered MTD techniques can provide a more adaptive and responsive defense compared to time-based techniques[52].

However, they require accurate and timely detection of relevant events, as well as careful design to avoid excessive overhead or instability due to frequent changes[51]. In practice, MTD systems often employ a combination of time-based and event-triggered techniques to achieve a balance between security, performance, and adaptability[48], [52]. **Hybrid-based** The hybrid-based approach combines elements of both time-based and event-based techniques[52]. In this approach, the system can be configured to perform regular, scheduled movements (time-based) while also responding to specific events or conditions (event-based). This hybrid approach aims to provide a balance between proactive and reactive defense mechanisms, leveraging the strengths of both techniques. The choice of timing function depends on various factors, such as the system's requirements, the nature of the threats, and the desired level of unpredictability and adaptability. In some cases, a combination of different timing functions may be employed to create a more comprehensive and effective MTD strategy.

2.3.5 Benefits and Challenges of MTD

This section outlines the key advantages and drawbacks of implementing Moving Target Defense (MTD) strategies. The benefits. The benefits of implementing Moving Target Defense (MTD) strategies include increased attack complexity for adversaries, resilience against zero-day exploits, disruption of the cyber kill chain, reduced attack surface, and an asymmetric advantage for defenders. However, MTD also presents challenges such as potential performance impact, management complexity, the need for robust evaluation frameworks, compatibility issues with legacy systems, and the risk of false positives.

Benefits

- Increased attack complexity and cost for adversaries[48], [51]–[54]
  - By continuously changing the attack surface, MTD forces attackers to invest more time and resources in reconnaissance and attack development, reducing the overall success rate of attacks[48].
  - It reduces the window of opportunity for attackers to exploit vulnerabilities before they become obsolete[48], [51].
- Resilience against zero-day exploits[55][48], [53], [56]
  - The dynamic nature of MTD makes it harder for attackers to rely on previously unknown vulnerabilities (zero-days)[48].
  - It provides proactive defense against new and emerging threats[56].
- Disruption of the cyber kill chain[51], [54], [57]
  - MTD disrupts the attacker's ability to gain a foothold, maintain persistence, and progress through the stages of an attack[51].
  - It forces attackers to deal with a constantly evolving and compromised target.

- Reduced attack surface and opportunities[48], [51], [58], [59]
  - By dynamically changing system configurations, MTD reduces the overall attack surface and potential entry points for attackers[48], [51].
- Asymmetric advantage for defenders[51], [56], [60], [61]
  - MTD reverses the traditional asymmetric advantage held by attackers in static environments[56].
  - It introduces uncertainty and complexity for attackers while providing defenders with a proactive approach[51].

Challenges

- Performance impact and overhead[48], [51], [52]
  - Certain MTD techniques, like continuous randomization and diversity, can introduce overhead and latency, potentially affecting system performance[51].
- Management complexity[48], [51], [52]
  - Coordinating and managing dynamic changes across multiple systems and networks can be challenging, requiring specialized tools and processes[51].
  - Increased complexity may lead to unintended vulnerabilities if not properly managed[52].
- Need for robust evaluation frameworks[48], [51], [52]
  - Measuring the effectiveness of MTD techniques and comparing different approaches can be difficult due to the dynamic nature of the systems[51].
  - Establishing baseline metrics and performance indicators is challenging[52].
- Compatibility issues[52]
  - MTD may not be compatible with legacy systems or applications that are not designed to support dynamic configurations or changes.
- Potential for false positives[52]
  - MTD may generate false positives, such as legitimate users being blocked from accessing the system, leading to user frustration and decreased productivity[52].

Chapter 3

Related Work

This chapter provides an overview of the related work in the field of Moving Target Defense (MTD) and its applications in Federated Learning (FL), Machine Learning (ML), and Blockchain. The chapter begins by discussing various security techniques employed in FL, followed by a detailed analysis of MTD in Centralized Federated Learning (CFL) and Decentralized Federated Learning (DFL). Furthermore, the chapter explores the applications of MTD in other domains, such as ML and Blockchain, highlighting the versatility and potential of MTD techniques in enhancing security and resilience against adversarial attacks.

3.1 Security Techniques in Federated Learning

Federated Learning (FL) has emerged as a promising approach for collaborative learning while preserving data privacy. However, FL systems face various security challenges, making it crucial to employ robust security techniques to protect the privacy and integrity of the learning process.

3.1.1 Homomorphic Encryption (HE)

HE allows computations to be performed directly on encrypted data without requiring access to the decryption key. In the context of federated learning, HE enables multiple parties to collaboratively train a machine learning model while keeping their raw training data private and encrypted. FedML-HE [62] introduces an optimized system for federated learning that utilizes homomorphic encryption (HE) to provide strong privacy guarantees during training. The authors demonstrate that their system significantly reduces the computational overhead associated with HE, especially for large foundation models like ResNet-50 and BERT, achieving up to 10x and 40x reduction in overhead respectively[1]. These optimizations make FedML-HE a promising solution for scalable and privacy-preserving federated learning deployments.

3.1.2 Differential Privacy (DP)

DP involves adding carefully calibrated noise to the model updates to mask the contribution of any individual client's data. When DP is applied to FL, it ensures that the trained model does not memorize or reveal private data of individual clients, even in the presence of adversarial attacks. [63] proposes a differentially private federated learning framework that provides theoretical privacy guarantees for client data while maintaining high model performance. The authors introduce two algorithms, Distributed Stochastic Gradient Descent with Differential Privacy (DSGD-DP) and Distributed Adam with Differential Privacy (DAdam-DP), which leverage differential privacy techniques like gradient clipping and random noise addition to prevent leakage of sensitive information during federated training. Through extensive experiments on benchmark datasets, they demonstrate that their approach achieves comparable accuracy to non-private federated learning while providing strong privacy protection, paving the way for more secure and privacy-preserving collaborative learning systems.

3.1.3 Byzantine-Robust Aggregation Methods

Byzantine-Robust Aggregation refers to a class of techniques used in federated learning to ensure that the global model can be trained correctly even in the presence of Byzantine clients, i.e., clients that behave arbitrarily or maliciously. These techniques aim to make the aggregation of model updates from clients robust to outliers, noise, and adversarial attacks.

Krum [64] is a popular Byzantine-robust aggregation rule used in federated learning to mitigate the impact of adversarial or faulty clients (workers). In each round of federated learning, the central server receives local model updates from the clients. Krum aims to select the most reliable update to update the global model, while filtering out outliers or malicious updates. The key idea of Krum is to compute a score for each local model update based on its Euclidean distances to the other updates. For each update vector, Krum calculates the sum of the squared Euclidean distances to its $n-f-2$ closest vectors, where n is the total number of clients and f is the maximum number of Byzantine clients tolerated. The update with the lowest score is then selected to update the global model. Krum provides theoretical guarantees for convergence under certain assumptions on the number of Byzantine clients (f) and the total number of clients (n), requiring $n \geq 2f + 3$ [65].

Coordinate-wise Median and Coordinate-wise Trimmed Mean [66] are two robust aggregation methods for distributed learning. The coordinate-wise median is computed by selecting the median value for each coordinate independently across all vectors, while the coordinate-wise β-trimmed mean removes the largest and smallest β fraction of values for each coordinate before computing the average. The paper proves that distributed gradient descent algorithms using these aggregation methods can achieve optimal statistical error rates under Byzantine attacks. The key difference is that median is more robust to extreme outliers but less efficient, while trimmed mean is more efficient but assumes evenly distributed outliers. The trimming parameter β allows for a trade-off between robustness and efficiency.

3.1.4 Blockchain Integration

Blockchain technology can be integrated with federated learning to enhance security and privacy. The key idea is to leverage the decentralized, immutable, and transparent nature of blockchain to address some of the challenges and vulnerabilities in traditional federated learning systems. In terms of DFL, Blockchain is a natural fit, as it aligns with the goal of creating a fully decentralized and secure learning environment.

[67] proposes a novel approach called FL-Block that integrates blockchain technology with federated learning to enhance privacy and security in fog computing environments. FL-Block leverages a distributed hash table and a proof-of-stake consensus mechanism to ensure efficient block generation and validation in the blockchain network. The authors present a system architecture and a detailed algorithm for FL-Block, which enables decentralized and privacy-preserving model training without relying on a central authority. The proposed approach is evaluated through simulations, demonstrating its effectiveness in terms of model accuracy, convergence speed, and resistance to various attacks.

[68] proposes a novel approach for generating synthetic data using federated learning. The authors argue that while deep learning algorithms require massive datasets for training, the diversity and expressiveness of the generated synthetic data are limited by the statistical properties of the available dataset within a single organization. To address this challenge, they introduce FedSyn, a framework that leverages federated learning to enable the generation of more diverse and generic synthetic data by utilizing datasets from multiple organizations without compromising data privacy. The proposed approach is evaluated through experiments, demonstrating its effectiveness in generating high-quality synthetic data while preserving the privacy of the underlying real data.

3.2 Moving Target Defense (MTD) in Federated Learning

Moving Target Defense (MTD) is a proactive defense strategy that introduces randomness and unpredictability into the system to thwart adversarial attacks. MTD techniques have shown potential in enhancing the security of FL systems.

3.2.1 MTD in Centralized Federated Learning (CFL)

In CFL, MTD techniques can be applied at the server level to protect the aggregation process and the global model. Examples of MTD techniques in CFL include dynamic aggregation algorithms, secure aggregation with MTD, and adaptive client selection. These techniques introduce randomness and diversity in the aggregation process, making it harder for attackers to manipulate the global model.

[8] proposes a novel approach to enhance the security of federated learning systems by applying the concept of MTD. The authors introduce an augmented dual-shuffle mechanism that dynamically changes the attack surface by shuffling both the client selection

and the model aggregation process. This MTD strategy aims to ensure the confidentiality, integrity, and availability (CIA-triad) of the federated learning system by making it more difficult for attackers to target specific clients or manipulate the global model. The proposed approach is evaluated through simulations and compared with existing methods, demonstrating its effectiveness in improving the security and robustness of federated learning against various attacks.

3.2.2 MTD in Decentralized Federated Learning (DFL)

In DFL, MTD techniques are implemented at the client level to protect the communication between clients and prevent malicious clients from disrupting the learning process. Examples of MTD techniques in DFL include random neighbor selection, dynamic communication channels, consensus protocols with MTD, and decentralized data storage with MTD.

[2] introduces an innovative solution to strengthen the security of decentralized federated learning (DFL) systems against communication threats, including eavesdropping and eclipse attacks. The authors leverage the concept of Moving Target Defense (MTD) to dynamically alter the communication patterns and network configurations in DFL, increasing the difficulty for attackers to target specific nodes or intercept sensitive information. The proposed security module employs both symmetric and asymmetric encryption techniques to safeguard the data and metadata of the participants. Additionally, it incorporates MTD techniques, such as random neighbor selection and IP/port switching, to enhance the unpredictability and resilience of the network. The experimental results demonstrate that the security module achieves an impressive average F1 score of 95% for the federated model while effectively mitigating the risks of eavesdropping, Man in the Middle attacks, network mapping, and eclipse attacks. However, the implementation of the security module does introduce moderate increases in CPU usage, network traffic, and RAM usage.

Voyager [31] proposes a novel aggregation protocol that uses a reactive MTD-based mechanism to mitigate poisoning attacks by dynamically changing the network topology. The paper analyzes the impact of network topology on the security risk of DFL, and designs a three-stage MTD mechanism that consists of an anomaly detector, a topology explorer, and a connection deployer. The paper evaluates Voyager on various datasets, network topologies, and attack scenarios, and compares it with other aggregation algorithms. The results show that Voyager is effective in reducing the impact of poisoning attacks without increasing the network overhead and resource consumption. However, this paper only focused on analyzing datasets in an IID setting. In the future, Voyager's performance is intended to be evaluated in non-IID scenarios through various experiments and proactive strategies for MTD will be also considered and implemented.

Both CFL and DFL can benefit from MTD techniques to enhance their security and robustness against various attacks, such as poisoning attacks, inference attacks, and communication threats. While the core principles of MTD remain the same, the decentralized nature of DFL introduces additional challenges in coordinating the MTD strategies across

the network. The specific implementation details and the trade-offs between security, performance, and complexity may differ based on the centralized or decentralized architecture of the federated learning system.

MTD for DFL and CFL is still a relatively new field, to gain additional insights and identify potential defensive techniques, this work expands the scope of analysis to examine how MTD has been applied successfully in other domains such as in Machine Learning(ML) and Internet of Things(IoT).

3.3 Moving Target Defense (MTD) in Other Domains

MTD techniques have found applications beyond federated learning, showcasing their versatility and potential in enhancing security and resilience against adversarial attacks in various domains.

3.3.1 MTD in Machine Learning

MTD is a proactive defense strategy that has been applied in the field of Machine Learning (ML) to enhance the security and robustness of ML systems against various attacks. The goal of MTD in ML is to create a dynamic and unpredictable environment for attackers by continuously changing the attack surface of the ML model or the training process.

[69] proposes a novel approach to enhance the robustness of neural networks against adversarial attacks by leveraging the concept of MTD. The authors introduce a dynamic strategy that continuously changes the model's architecture and parameters during inference, making it harder for attackers to craft effective adversarial examples. The proposed MTD technique is evaluated on various datasets and compared with existing defense mechanisms, demonstrating its effectiveness in improving the model's resilience against both white-box and black-box attacks. The results show that the MTD approach can significantly reduce the success rate of adversarial attacks while maintaining the model's performance on benign inputs, providing a promising direction for enhancing the security of machine learning systems.

[70] proposes a novel Moving Target Defense (MTD) strategy called Morphence to enhance the robustness of deep learning models against adversarial attacks. The key idea of Morphence is to generate a diverse set of student models by applying random perturbations to the weights of a base model, and then randomly selecting one of the student models for each inference request. The authors evaluate the effectiveness of Morphence on two datasets, CIFAR-10 and MNIST, and demonstrate that it significantly reduces the success rate of adversarial attacks while maintaining high accuracy on benign inputs. The results also show that increasing the noise scale used for weight perturbations leads to a decrease in the transferability rate of adversarial examples across student models, indicating the generation of more diverse models.

[71] proposes EI-MTD, which is a dynamic defense mechanism proposed by the paper to address the challenges of limited resources, transferability, and static models in Edge Intelligence (EI) settings. Edge intelligence is the application of deep learning models on edge nodes, such as edge devices and edge servers, to provide real-time and intelligent services. [72] The paper proposes a three-stage framework that includes a knowledge distillation stage, a Bayesian Stackelberg game stage, and a dynamic defense stage. The framework aims to prevent the adversary from finding a proper substitute model, reduce the transferability of adversarial attacks without compromising accuracy, and defend against adversarial examples with limited resources on edge nodes.

[73] proposes MT-MTD, a moving target defense framework to protect deep neural networks (DNNs) against backdoor/Trojaning attacks in edge AI environments. MT-MTD divides the training data into multiple dimensions, has an untrusted attacker train each dimension which may contain backdoors, and has the defender randomly select dimensions and reach consensus on the output. Through repeated retraining rounds and adjusting dimension weights based on a signaling game between the attacker and defender, MT-MTD increases the attack cost and unpredictability for the attacker, while maintaining high accuracy for the defender. Experiments on GTSRB and ImageNet datasets show MT-MTD can quickly converge to 90% accuracy even under strong attacks, by using a random dimension selection probability. Overall, MT-MTD provides an active, dynamic defense against backdoor attacks in edge AI systems, significantly increasing attacker cost and difficulty.

[74] proposes a novel approach called MTDeep to enhance the robustness of neural networks against adversarial attacks by leveraging the concept of Moving Target Defense (MTD). The authors introduce a dynamic strategy that continuously changes the model's architecture and parameters during inference, making it harder for attackers to craft effective adversarial examples. The proposed MTD technique is evaluated on various datasets and compared with existing defense mechanisms, demonstrating its effectiveness in improving the model's resilience against both white-box and black-box attacks. The results show that the MTD approach can significantly reduce the success rate of adversarial attacks while maintaining the model's performance on benign inputs, providing a promising direction for enhancing the security of machine learning systems.

3.3.2 MTD in Internet of Things

[75] provides a comprehensive review of Moving Target Defense (MTD) techniques specifically designed for the Internet of Things (IoT) domain, offering valuable insights into the current landscape of MTD techniques for IoT, highlighting the progress made so far and the areas that require further research and improvement.

[76] proposes a novel lightweight MTD technique called μM6TD to enhance the security of IoT devices against network-based attacks . Micro-MTD leverages the vast address space of IPv6 to frequently rotate the network addresses of IoT devices, making it harder for attackers to target specific devices or maintain a persistent presence in the network. The authors present the design and implementation of Micro-MTD, which includes a lightweight address rotation mechanism and a secure communication protocol for IoT

devices. The evaluation results demonstrate that Micro-MTD effectively mitigates various network-based attacks, such as scanning, sniffing, and DoS attacks, while introducing minimal overhead on the resource-constrained IoT devices.

[77] introduce 6HOP, which exploits the vast address space of IPv6 to defend against reconnaissance attacks, address-based correlation, and denial-of-service attacks. 6HOP is designed to be lightweight in operation and requires minimal administration overhead, making it suitable for resource-constrained IoT devices. By frequently changing the IPv6 addresses of IoT devices, 6HOP makes it harder for attackers to locate and target specific devices, thus providing an effective protection mechanism for the IoT ecosystem.

[78] presents FeDef, a federated and cooperative framework for deploying Moving Target Defense (MTD) techniques on resource-constrained IoT devices affected by command and control-based malware. FeDef combines reactive and proactive MTD approaches, where an infected device reactively changes its IP address to disrupt communication with the command and control server, while also proactively notifying other devices to change their Telnet service port to prevent malware spreading. The performance of FeDef was evaluated in a simulated environment with devices infected by the Bashlite malware, demonstrating improvements in overall infection time, service disruption, and resource consumption compared to purely reactive or proactive non-cooperative scenarios. The results show FeDef can be implemented on resource-constrained devices with minimal impact on network and service availability.

3.4 Motivation

This chapter provided a comprehensive overview of the related work in the field of MTD and its applications in FL, ML, and IoT. The chapter discussed various security techniques employed in FL, followed by a detailed analysis of MTD in CFL and DFL. Furthermore, the chapter explored the applications of MTD in other domains, highlighting the versatility and potential of MTD techniques in enhancing security and resilience against adversarial attacks. Table 3.1 compares various MTD techniques implemented in different domains, including CFL, DFL, ML and IoT, highlighting the specific methods used, whether they are proactive or reactive, and the types of attacks they defend against.

It's important to note that the field of MTD in FL is still evolving, especially DFL. While a few studies have explored the application of MTD in DFL to mitigate communication threats and improve model robustness, there is still a significant lack of comprehensive frameworks and practical implementations that effectively integrate MTD strategies into DFL architectures to mitigate poisoning attacks. To fill the gap, this work aims to provide a comprehensive framework for the MTD method of DFL by exploring its implementation from multiple perspectives. This will be followed by the design of an adaptive MTD mechanism and the evaluation of its performance against diverse poisoning attack scenarios, thereby promoting the development and innovation of this field.

Table 3.1: Comparison of MTD techniques in various domains

Reference	Domain	Year	Method	When	Attack Defended Against
[8]	CFL	2021	Hybrid: Shuffling (dual-shuffle), encryption	Proactive	Label Flipping; Additive Gaussian noise; Back-door Attacks
[2]	DFL	2023	Hybrid: Shuffling (IP, port), encryption, random neighbor selection	Both	Eavesdropping; Man in the Middle; Network mapping; Eclipse attacks
[31]	DFL	2024	Hybrid: Shuffling (network topology), anomaly detection	Reactive	Untargeted label flipping; Model poisoning
[69]	ML	2019	Diversity: Ensemble of models	Proactive	Adversarial attacks
[70]	ML	2021	Diversity: Model pool generation, detection	Reactive	Adversarial attacks
[71]	ML (Edge)	2020	Hybrid: Shuffling (model parameters), detection	Proactive	Adversarial attacks
[73]	ML (Edge)	2021	Hybrid: Multi-training, model selection	Proactive	Trojaning attacks
[74]	ML (Deep Learning)	2017	Diversity: Ensemble of models	Proactive	Adversarial attacks
[76]	IoT	2017	Shuffling: IPv6 address hopping	Proactive	Network-based attacks
[77]	IoT	2017	Shuffling: IPv6 address hopping	Proactive	Network-based attacks
[78]	IoT	2023	Hybrid: Shuffling (IP, port), diversity (network, data, runtime environment)	Both	Command and control-based malware
This work	DFL	2024	Shuffling: Randomized Aggregation Functions	Both	Model Poisoning

Chapter 4

Defense Design

This chapter presents the design and architecture of the proposed MTD-based defense mechanism for DFL.

4.1 MTD Techniques in FL

Figure 2.7 provides a taxonomy of various MTD techniques, including attacking surfaces, types and the timeliness. Following the taxonomy, this work aims to create a comprehensive framework for understanding the MTD method of DFL. This framework will be developed by exploring various implementation perspectives, including shuffling, diversity, redundancy, hybrid techniques and possible attacking surfaces in DFL.

4.1.1 Shuffling

Shuffling techniques introduce dynamism into system and network configurations through rearrangement or randomization. This section delves into the specific ways these configurations can be altered. Table 4.1 shows different types of shuffling techniques that can be used as moving target defense strategies in DFL. The table also highlights the three fundamental questions in MTD: WHAT, HOW and WHEN to move.

1. IP/port Shuffling: This technique involves randomly changing the IP addresses and ports used for communication between nodes in the DFL network. It helps introduce unpredictability in the communication patterns, making it harder for adversaries to launch attacks.

2. Network Topology Shuffling: This shuffling method dynamically modifies the network topology by changing the connections between nodes over time. It prevents adversaries from exploiting static network topologies and communication patterns.

Type [HOW]	Main Techniques [WHAT]	Pros	Cons	Proactive or Reactive [WHEN]	CFL or DFL
Shuffling	IP/port Shuffling	Leveraging legacy devices or technologies; Providing **affordable economical** defense	Potential high **communication cost or service interruption** if not executed adaptively; **Limited by the inherent vulnerabilities of the existing technologies** used or shuffling spaces	Both	Both
	Network Topology Shuffling			Both	Both
	Aggregator Randomization			Both	DFL
	Randomized Aggregation Functions			Both	Both
	Node Role			Both	DFL
	Encryption Key Shuffling			Both	Both
	VM Instances Migration			Both	Both
	Communication Link Shuffling			Both	Both

Figure 4.1: A Summary of Pros and Cons of MTD Techniques-Shuffling

3. Aggregator Randomization: In this technique, different nodes are randomly selected to act as aggregators for model parameter aggregation during each round of the federated learning process. This randomization makes it difficult for adversaries to target specific aggregator nodes.

4. Randomized Aggregation Functions: Instead of using a fixed aggregation function (e.g., FedAvg, Krum, Median, TrimmedMean) for combining model parameters from different nodes, this technique employs different randomized functions for aggregation. This unpredictability in the aggregation process enhances security.

5. Node Role Shuffling: The roles of nodes (trainer, aggregator, proxy, idle) are dynamically changed during the learning process. This shuffling prevents adversaries from exploiting vulnerabilities associated with specific node roles.

6. Encryption Key Shuffling: This technique periodically changes the encryption keys used for secure communication between nodes, making it harder for adversaries to decrypt intercepted messages.

7. VM Instances Migration: In this shuffling method, the virtual machine instances hosting the nodes are moved to different physical machines over time, introducing unpredictability in the physical infrastructure.

8. Communication Link Shuffling: The communication links between nodes are changed dynamically, preventing adversaries from exploiting static communication patterns.

Pros and Cons: On the positive side, shuffling techniques can introduce unpredictability and increase the complexity for adversaries to launch successful attacks against the DFL system. By dynamically changing the attack surface, these techniques can invalidate the intelligence gathered by attackers and deplete their resources. By dynamically changing the attack surface, these techniques can invalidate the intelligence gathered by attackers and deplete their resources. The shuffling techniques are jighly compatible with legacy devices and technologies, allowing immediate applicability. They can also work with

Type [HOW]	Main Techniques [WHAT]	Pros	Cons	Proactive or Reactive[WHEN]	CFL or DFL
Diversity	Programming Diversity	Leveraging legacy devices or technologies; Working well with shuffling-based MTD to **double the effect**.	Limited by the inherent vulnerabilities of the existing technologies or **a number of software alternatives**	Both	Both
	Model Diversity			Both	Both
	Data Diversity			Both	Both
	Node Type Diversity			Both	Both

Figure 4.2: A Summary of Pros and Cons of MTD Techniques-Diversity

existing technologies without developing a new security mechanism, reducing development costs and resources.

However, the main drawback is the potential high communication cost or service interruption if not executed adaptively. Frequent shuffling can lead to increased network overhead, latency, and potential disruptions in the learning process, especially in resource-constrained or bandwidth-limited DFL scenarios. Therefore, a careful balance must be struck between the security benefits of shuffling and the associated performance trade-offs to ensure efficient and reliable DFL operations. Further, the security achieved is limited by the vulnerabilities of the existing legacy devices/technologies that shuffling relies on. If those technologies are not robust enough against attacks, the shuffling technique's effectiveness is diminished. The size of the shuffling spaces (e.g., number of virtual IPs that can be assigned to a real IP) is also critical for enhancing security. Limited shuffling spaces can reduce the technique's effectiveness.

CFL or DFL: Most of the shuffling techniques listed can be applied to both CFL and Dfl except for two techniques: Aggregator Randomization and Node Role Shuffling. In Aggregator Randomization, the technique is specific to DFL scenarios, as it involves randomly selecting different nodes to act as aggregators for model parameter aggregation, which is a characteristic of decentralized architectures. Similar to Aggregator Randomization, Node Role Shuffling is specific to DFL scenarios, as it involves dynamically changing the roles (trainer, aggregator, proxy, idle) of nodes during the learning process, which is a characteristic of decentralized architectures.

4.1.2 Diversity

Table 4.2 shows different types of diversity techniques that can be used as moving target defense strategies in DFL.

1. Programming Diversity refers to leveraging legacy devices or technologies and working well with shuffling-based MTD techniques to enhance the effect. However, the PDF does not provide specific examples of programming diversity techniques.

2. Model Diversity involves using different types of machine learning models (e.g. MLP, CNN, neural networks, decision trees) across the nodes in the federation. The idea is that having diverse model architectures across nodes can improve robustness against attacks.

3. Data Diversity aims to partition the training data in different ways across nodes, such as using horizontal federated learning (different samples), vertical federated learning (different features), or transfer federated learning (limited shared samples/features). Having diverse data distributions makes it harder for an attacker to exploit.

4. Node Type Diversity refers to having a heterogeneous mix of hardware configurations (Virtual participants, Real devices or External datacenter). The diversity of node types increases complexity for potential attackers.

Pros and Cons: On the positive side, diversity MTD techniques can increase the complexity and unpredictability for potential attackers, making it harder to exploit vulnerabilities or launch successful attacks against the DFL system. This added diversity introduces uncertainty and dynamism, disrupting the attacker's asymmetric advantage. By leveraging existing technologies can make implementation easier and more cost-effective compared to developing entirely new systems. Combining diversity-based and shuffling-based MTD techniques can potentially "double the effectiveness" by introducing multiple layers of moving targets and unpredictability for attackers.

However, a key con is that leveraging legacy devices or technologies with diverse configurations could be limited by the inherent vulnerabilities or constraints of those existing systems, potentially introducing new attack surfaces or compatibility issues within the federated environment. Maintaining and orchestrating such diversity across participants may also increase management overhead and complexity for the DFL implementation. Further, if the underlying existing technologies themselves have high vulnerabilities, simply adding an MTD layer on top may not provide significantly better security, despite the additional defense costs incurred. Like the shuffling techniques, the degree of diversity available is also a limiting factor. If there are not many diverse alternatives or versions to choose from, the potential security enhancement will be inherently limited.

CFL or DFL: In principle, the core concepts of data, model, node, and device diversity seem applicable to both CFL and DFL. However, the specific implementation details and considerations for applying these diversity techniques may differ between CFL and DFL due to their architectural differences in terms of communication patterns, aggregation mechanisms, etc.

4.1.3 Redundancy

By implementing redundancy techniques, DFL systems can enhance their overall reliability, fault tolerance, and resilience against various types of failures, such as node failures, communication disruptions, or malicious attacks. Table 4.2 shows different types of redundancy techniques that can be used as moving target defense strategies in DFL.

1. Data Redundancy: This technique involves creating and maintaining multiple copies or replicas of the same data across different nodes or locations within the DFL system. By having redundant data copies, the system can continue operating even

Type [HOW]	Main Techniques [WHAT]	Pros	Cons	Proactive or Reactive [WHEN]	CFL or DFL
Redundancy	Data Redundancy	**Higher reliability and service availability**; **easily combined** with shuffling or diversity to significantly secure a system and ensure service availability	**Additional cost to move / setup** additional system components (e.g., servers or routing paths); If not properly executed, it **increases attack surface**	Both	Both
	Model Redundancy			Both	Both
	Redundant Aggregation Nodes			Both	Both
	Redundant Communicant Paths				
	Federated Controller Redundancy			Both	Both
	Redundant Local Updates			Both	Both

Figure 4.3: A Summary of Pros and Cons of MTD Techniques-Redundancy

if one or more nodes fail or become unavailable, as the data can be accessed from the remaining replicas.

2. Model Redundancy: In this approach, multiple instances or replicas of the same machine learning model are trained and maintained across different nodes in the DFL network. This redundancy ensures that if one model instance fails or is compromised, the system can rely on the other replicated models to continue providing the required functionality.

3. Redundant Aggregation Nodes: This technique involves having multiple nodes designated as aggregators within the DFL system. These redundant aggregation nodes are responsible for collecting and aggregating the local model updates from other nodes during the federated learning process. If one aggregation node fails, the others can take over the aggregation task, ensuring continuity of the learning process.

4. Redundant Communicant Paths: In this case, redundancy is introduced at the communication level by establishing multiple communication paths or routes between nodes in the DFL network. If one communication path fails or is disrupted, the nodes can switch to alternative redundant paths to maintain connectivity and continue exchanging model updates or other data.

5. Federated Controller Redundancy: This technique involves having multiple federated controllers or coordinating nodes within the DL system. These redundant controllers are responsible for managing and orchestrating the federated learning process, including tasks such as node selection, model aggregation, and synchronization. If one controller fails, the others can take over its responsibilities, ensuring the overall system's resilience.

6. Redundant Local Updates: In this approach, each node in the DFL system maintains redundant copies or versions of its local model updates before sharing them with other nodes or the aggregator. This redundancy helps mitigate the risk of data loss or corruption during the communication or aggregation process.

Pros and Cons: Redundancy MTD techniques provide multiple replicas of system or network components offering the same functionality at the network or application layer.

The key pros are higher reliability and service availability, as the system can continue operating even if one or more nodes fail or become unavailable due to the redundant replicas. It also can also be combined with shuffling or diversity to significantly secure a system and ensure service availability.

However, the cons include additional costs to move or setup additional system components like servers or routing paths. If not properly executed, it increases the attack surface. There is also a risk that if not implemented correctly, redundancy can lead to inconsistencies across different network segments, negatively impacting the accuracy of federated models for certain participant groups

CFL or DFL: While most redundancy techniques can be applied to both DFL and CFL, they may have different levels of importance and impact depending on the architecture.

Data Redundancy, Model Redundancy, Redundant Aggregation Nodes, and Redundant Local Updates can be applied to both DFL and CFL architectures. These techniques aim to create redundant copies or replicas of data, models, aggregation nodes, or local updates to enhance reliability, fault tolerance, and resilience against failures or attacks.

Redundant Communicant Paths may be more relevant and beneficial in DFL architectures compared to CFL. In DFL, there is no central server, and nodes communicate directly with each other. Having redundant communication paths can help maintain connectivity and ensure model parameter exchanges even if some paths or nodes fail. In CFL, the communication is primarily between clients and the central server, so redundant paths may not be as crucial. Federated Controller Redundancy is more applicable to CFL architectures, where there is a central coordinating entity (the server). Having redundant federated controllers can prevent single points of failure and ensure continuity of the federated learning process. In DFL, there is no central controller, and the coordination is decentralized among the nodes, so this technique may not be as relevant.

4.1.4 Hybrid

Hybrid MTD coordinates the use of multiple MTD techniques (shuffling, diversity, and redundancy) to provide a more comprehensive defense. Figure 4.4 illustrates the relationships between three key principles of MTD techniques: shuffling, diversity, and redundancy.

”Shuffling” aims to improve performance and efficiency by continuously changing the attack surface, making it harder for attackers to exploit vulnerabilities. ”Diversity” contributes to resilience and robustness by introducing heterogeneity into the system components, reducing the risk of widespread compromise from a single vulnerability. ”Redundancy” enhances reliability and availability by providing redundant components or resources, allowing the system to continue operating even if some parts are compromised. Their combined application can improve the performance and efficiency of cybersecurity systems by increasing complexity and unpredictability for attackers.

Hybrid MTD approaches that integrate multiple techniques can provide enhanced security benefits compared to single-technique solutions. Combining diversity or redundancy

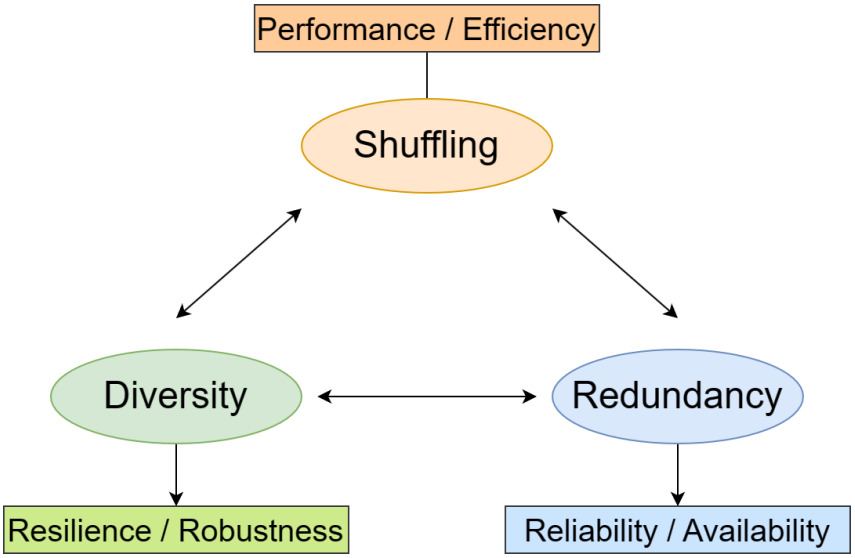


Figure 4.4: Relationships between Shuffling, Diversity and Redundancy

with shuffling significantly improves protection while potentially reducing defense costs and service disruptions. For instance, MTD that incorporates both shuffling and diversity may require less frequent shuffling compared to a shuffling-only approach, as the diversity of system components increases the difficulty for attackers to identify vulnerabilities. Additionally, redundancy improves system availability, resulting in higher service quality and fewer interruptions for users. However, hybrid MTD may introduce a larger attack surface compared to a single MTD technique. It also involves additional overhead and complexity in integrating multiple techniques into a unified solution, leading to a complex, multi-objective optimization problem with various operational constraints

4.2 Defense Design

Having explored the range of MTD techniques in DFL, this work will now design a MTD technique to strengthen the security and resilience of DFL systems against poisoning attacks. Randomized Aggregation Functions from Shuffling MTD techniques can serve as a good starting point as they introduce unpredictability in the aggregation process, making it harder for adversaries to predict and manipulate the aggregated model updates.

4.2.1 Dynamic Aggregation Functions

The idea for Randomized Aggregation Functions, or Dynamic Aggregation Functions can be summarized as below:

1. The system maintains a pool of aggregation functions, including FedAvg, Krum, Median, and Trimmed Mean. The node can dynamically switch between these functions during the federated learning process.

2. Dynamic aggregation is triggered in two modes:

   - Proactive mode: At the start of each training round, if the node itself is not malicious, then the node randomly selects a new aggregation function from the pool to use for this round.
   - Reactive mode: If malicious nodes are detected and the node itself is not malicious, it takes the following actions:
     - First, it broadcasts the list of detected malicious nodes to other nodes in the network.
     - Second, it switches to a randomly selected aggregation function (excluding the current one) and excludes the malicious nodes from the aggregation process.

   If a non-malicious node receives messages about malicious nodes in the network, it will:

     - Randomly select a new aggregation function (excluding the current one) and exclude the malicious nodes from the aggregation process.

In summary, dynamic aggregation allows the node to adaptively switch the aggregation function during the federated learning process, either proactively to improve robustness and adaptability or reactively in response to detected malicious nodes. This enhances the overall security and resilience of the federated learning system. This flowchart 4.5 illustrates the dynamic aggregation process. It shows the steps involved in determining whether to use proactive or reactive dynamic aggregation, and how the aggregation function is dynamically changed based on the selected mode.

4.2.2 Key Components

1. **Aggregation Function Pool**: To enable dynamic adaptation during federated learning, the system maintains a pool of aggregation functions (FedAvg, Krum, Median, Trimmed Mean). The node can randomly select and switch between these functions at runtime. The specific function selection algorithm is detailed in Algorithm 1.

2. **Malicious Nodes Detection**: To find out which nodes are malicious, [31] proposes a Anomaly Detector module to identify any abnormality within these models by using reputation. Reputation plays a key role in determining the influence of nodes' contributions during aggregation. Nodes with a good reputation have their inputs weighted more heavily, giving more importance to reputable participants. While those with higher performance have their reputation increase, nodes with lower performance have their reputation decrease. The Anomaly Detector computes the layer-wise cosine similarity(reputation) between the local model m_i and the received models m_j in the shared models $\mathbb{M}'$ in the DFL network. If any of these similarity scores exceeds the threshold κ_s, the process of reactive dynamic aggregation is triggered. The Anomaly Detector Algorithm is depicted in Algorithm 2 and Layer-wise Cosine Similarity in Algorithm 3

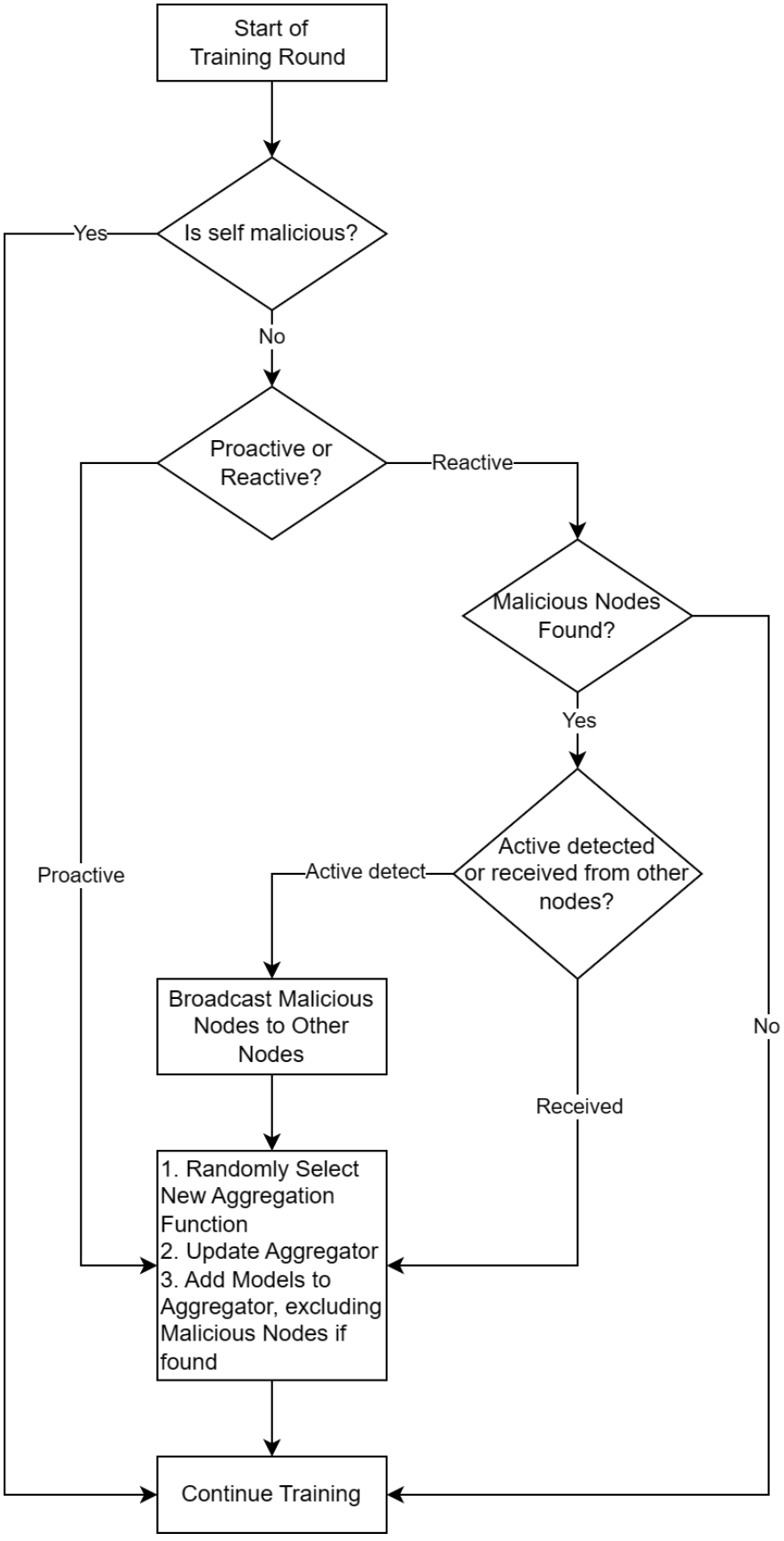


Figure 4.5: Dynamic Aggregation Flowchart

Algorithm 1 Random Aggregation Function Selection

Require: $\mathcal{A}$: aggregation functions pool, f: Current aggregation function of the node, f': Selected aggregation function
Ensure: $f' \neq f$
$\mathcal{A} \leftarrow$ "$Krum$","$Median$","$TrimmedMean$" ▷ Set of aggregation functions
if ∄ self.aggregator **then**
    $f \leftarrow \emptyset$ ▷ Current aggregation function
else
    $f \leftarrow$ self.aggregator
end if
$f' \leftarrow f$
while $f' = f$ **do**
    $f' \leftarrow$ random_choice($\mathcal{A}$)
end while
return f'

Algorithm 2 Anomaly Detector Algorithm [31]

Require: $\mathbb{M}'_i$: Neighbor models, m_i: Local model, κ_s: Similarity threshold
Initialize Triggering message $t \leftarrow 0$
for m_j in $\mathbb{M}'_i$ **do**
    $s_{ij} \leftarrow CosSim(m_i, m_j)$
    if $s_{ij} \geq r_s$ **then**
        $t \leftarrow 1$
    end if
end for
return t

Algorithm 3 Layer-wise Cosine Similarity [31]

Require: m_i: Local model, m_j: Neighbor model.
for l_i, l_j in m_i, m_j **do**
    $s_{ij} \leftarrow s_{ij} + \frac{l_i \cdot l_j}{\|l_i\|\|l_j\|}$
end for
return s_{ij}

3. **Dynamic Aggregator**: The Dynamic Aggregator is a core component that enables adaptability and security by randomly selecting a new aggregation function from the Aggregation Function Pool, updating the node's aggregator, and filtering out models from malicious nodes. The algorithm is explained in Algorithm 4

Algorithm 4 Dynamic Aggregator

Require: Aggregated models and weights $\mathbb{M}$, Malicious nodes N_m
Ensure: Updated Aggregator

$A_{\text{target}} \leftarrow$ Random Aggregation Function Selection ▷ New Aggregation Function
$A_{\text{current}} \leftarrow A_{\text{target}}$ ▷ Change the aggregator
$A_{\text{current}}.set\_nodes\_to\_aggregate(self.\_train\_set)$ ▷ Set Nodes
$A_{\text{current}}.set\_round(self.round)$ ▷ Set Round
for $s \in \mathbb{M}.keys()$ **do** ▷ Iterate over aggregated models (keys of M)
    $sublist \leftarrow s.split()$ ▷ Split nodes in string s
    $(m_s, w_s) \leftarrow M[s]$ ▷ Get model and weights
    for $n \in sublist$ **do** ▷ Iterate over nodes in sublist
        if $n \notin N_m$ **then** ▷ Not in malicious nodes
            $A_{\text{current}}.add\_model(m_s, \{n\}, w_s)$ ▷ Add Model
        end if
    end for
end for

Chapter 5

Implementation

5.1 Fedstellar Framework

Fedstellar[79] is an innovative, open-source platform[1] implemented in Python that enables the creation of centralized and decentralized FL architectures. It provides a standardized approach for developing, deploying, and managing FL applications while offering features for data management, model management, and performance monitoring. The architecture of Fedstellar consists of three main components:

- Frontend: A user-friendly frontend that allows for easy experiment setup and monitoring. It provides an intuitive interface for users to design and oversee learning scenarios.
- Controller: The controller serves as the orchestration center of the platform. It interprets user commands from the frontend, manages the entire federated scenario, assigns learning algorithms and datasets, and configures network topologies to ensure an efficient federated learning process.
- Core: The core component is deployed on each device participating in the federation. It is responsible for executing the federated learning tasks, including model training, data preprocessing, secure communication among devices, and storage of the federated models. The core also supervises the calculation of performance metrics and conveys this information back to the frontend for monitoring.

Docker is used to manage the various components of Fedstellar, including the frontend, controller, and participants. In the learning process, the user first defines the scenario configuration through the frontend, including parameters such as the number of rounds. The controller then loads this configuration and starts all participating nodes. Fedstellar assigns one of three roles to each node: Aggregators, which train local models on their own data and aggregate model updates from other nodes; Trainers, which only train local models and distribute updates without performing aggregation; and Proxies, which simply

[1] https://github.com/enriquetomasmb/fedstellar

relay model updates between nodes.Docker is used to manage the various components of Fedstellar, including the frontend, controller, and participants. After deployment, each node operates independently, training on its local dataset. Upon completing a training round, nodes send their updated parameters to their neighbors and wait to receive updates in return or until a timeout occurs. The process then moves to the aggregation phase and the next round, iterating until the specified number of rounds is completed.

Fedstellar also brings monitoring and analysis features. Fedstellar employs a customized implementation of TensorBoard, optimized for handling simultaneous metric updates from numerous devices, providing real-time insights into model performance, resource utilization, and communication status across the federated network. The platform also maintains log files for troubleshooting and replicating scenarios.

Worth to mention is that the latest version of Fedstellar, Fedstellar 2.0.0 introduces notable features for simulating and mitigating security threats in federated learning. The platform now allows users to add attack and defense mechanisms to the learning process. In the Frontend, users have the option to include malicious participants and simulate various poisoning attacks, such as targeted and untargeted data poisoning (e.g., label flipping, sample poisoning) and model poisoning. To counter these threats, [31] adds a reactive MTD mechanism. This approach dynamically alters the topology of the nodes to mitigate the impact of poisoning attacks. To achieve this, [31] incorporates a reputation system that calculates a reputation score for each node, aiding in the detection of malicious participants, as described in Algorithm 2 and 3.

5.2 Adaption of Dynamic Aggregation Functions

To adapt the design of Dynamic Aggregation Functions to Fedstellar Framework, this work adds a module to Fedstellar. In the Frontend, additional MTD selection and definition options to define MTD strategies are offered. Then the Controller receives instructions from the Frontend and deploys the entire federated scenario according to configuration. The Core encompasses the MTD Module. The MTD Module5.1 consists of the following sub-components:

- Reputation System: This utilises the Reputation System implemented by [31] in Fedstellar. It calculates and maintains reputation scores for the participating nodes/devices in the federated learning system. The reputation scores help identify and mitigate the impact of malicious or underperforming nodes during the model aggregation process. In this work, the threshold of cossim is set to 0.95 and the avgloss threshold is set to 0.4.

- Aggregation Function Pool: This is the implementation of Algorithm 1. The system maintains a pool of aggregation functions (FedAvg, Krum, Median, Trimmed Mean). The node can randomly select and switch between these functions at runtime.

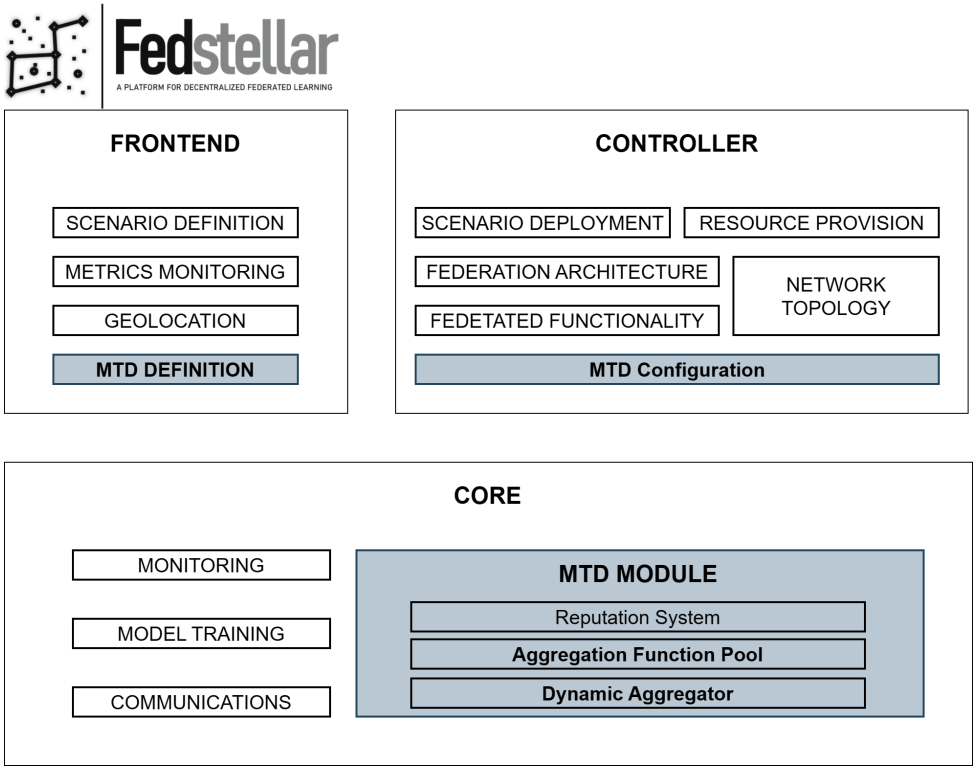


Figure 5.1: Architecture of the MTD Module in Fedstellar

- Dynamic Aggregator: This is the implementation of Algorithm 4. This is a core component that enables adaptability and security by randomly selecting a new aggregation function from the Aggregation Function Pool, updating the node's aggregator, and filtering out models from malicious nodes.

5.2.1 Adaption of Frontend and Controller

5.2 shows the user interface for deploying and configuring scenarios in the Fedstellar platform. When "Advanced mode" button, the user will have more configuration options such as "Advanced Topology", "Robustness" and "Defense", etc.

This work adds more options in the "Defense" configuration. Figure 5.3. The codes are implemented in file `deployment.html`, `participant.json.example` and `app.py`.

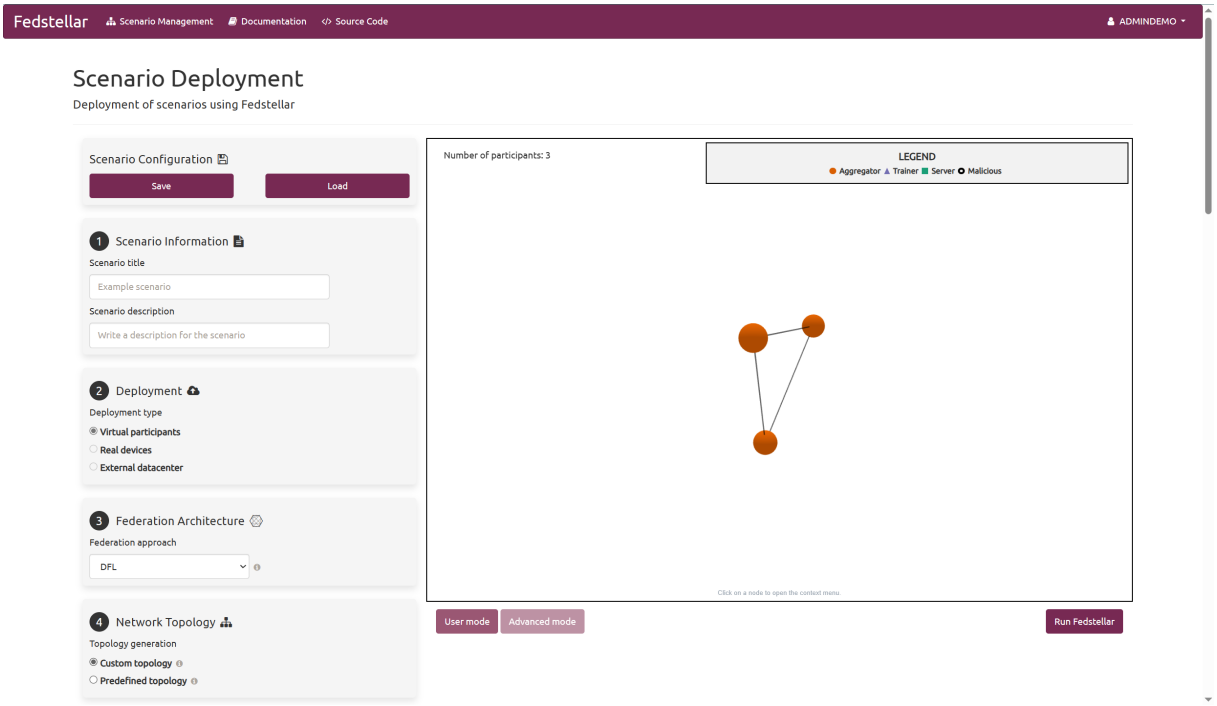


Figure 5.2: Fedstellar Frontend Interface

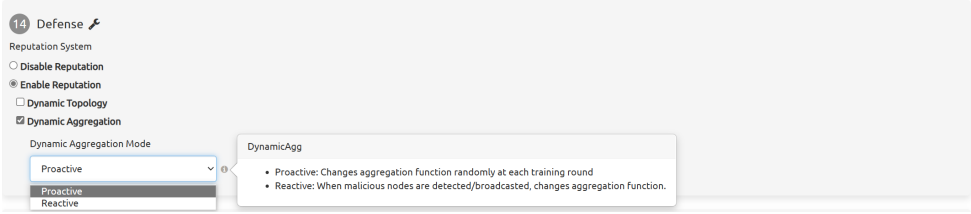


Figure 5.3: Fedstellar MTD Configuration

`deployment.html` modifies the frontend webpage.

```
data["is_dynamic_aggregation"] = document.getElementById("
   dynamic-aggregation-btn").checked ? true : false
data["dynamic_aggregation_mode"] = document.getElementById("
   dynamic-aggregation-btn").checked ? document.
   getElementById("dynamicAggregationModeSelect").value :
   false
document.getElementById("dynamic-aggregation-btn").checked =
   data["is_dynamic_aggregation"];
document.getElementById("dynamicAggregationModeSelect").value
    = data["dynamic_aggregation_mode"];

<div id="dynamic-aggregation-mode" style="display:␣none;␣
   margin-left:␣20px">
    <h5>Dynamic Aggregation Mode</h5>
    <select class="form-control" id="
       dynamicAggregationModeSelect" name="
       dynamicAggregationMode"
        style="display:␣inline;␣width:␣20%">
        <option selected>Proactive</option>
        <option>Reactive</option>
    </select>
    <small id="dynamicAggHelp" class="form-text␣text-muted">
        <i id="dynamicAggHelpIcon" class="fa␣fa-info-circle"
           ></i>
    </small>
</div>
```

Listing 5.1: deployment.html

`participant.json.example` parses paramaters from the frontend.

```
"defense_args": {
    "with_reputation": false,
    "is_dynamic_topology": false,
    "is_dynamic_aggregation": false,
    "target_aggregation": false,
    "dynamic_aggregation_mode": false
}
```

Listing 5.2: participant.json.example

`app.py` handles the scenario deployment with parameters from the frontend.

```
def fedstellar_scenario_deployment_run():
    ...
    participant_config["defense_args"]["
       is_dynamic_aggregation"] = data["
       is_dynamic_aggregation"]
```

```
4       participant_config["defense_args"]["
            dynamic_aggregation_mode"] = data["
            dynamic_aggregation_mode"]
```

Listing 5.3: app.py

5.2.2 Adaption of Controller

To implement Algorithm 1 and 4, `node.py` has been modified.

`__randomly_select_aggregation_function()` Randomly selects an aggregation function from a pool of aggregation functions. Krum, Median, and TrimmedMean are selected as candidate aggregation functions in the pool. These are common choices for Byzantine-Robust Aggregation, which aims to ensure accurate data aggregation even in the presence of faulty or malicious nodes (Byzantine faults) in distributed systems like federated learning. 3.1.3

```
1  def __randomly_select_aggregation_function(self):
2      # Define the pool of aggregation functions
3      aggregation_functions = ["Krum","Median", "TrimmedMean"]
4      # Get the current aggregation function
5      if not hasattr(self, 'aggregator'):
6          current_aggregation_function_name = None
7      else:
8          current_aggregation_function_name = self.
               aggregator_name
9      # Randomly select an aggregation function other than the
           current one
10     selected_aggregation_function_name =
           current_aggregation_function_name
11     while selected_aggregation_function_name ==
           current_aggregation_function_name:
12         selected_aggregation_function_name = random.choice(
               aggregation_functions)
13     # Create the aggregation function based on the randomly
           chosen name
14     if selected_aggregation_function_name == "Krum":
15         selected_aggregation_function = Krum(node_name=self.
               get_name(), config=self.config)
16         self.aggregator_name = "Krum"
17     elif selected_aggregation_function_name == "Median":
18         selected_aggregation_function = Median(node_name=self
               .get_name(), config=self.config)
19         self.aggregator_name = "Median"
20     elif selected_aggregation_function_name == "TrimmedMean":
21         selected_aggregation_function = TrimmedMean(node_name
               =self.get_name(), config=self.config)
```

```
22          self.aggregator_name = "TrimmedMean"
23      return selected_aggregation_function
```

Listing 5.4: __randomly_select_aggregation_function()

`__dynamic_aggregator()` Dynamically selects an aggregation function, updates the aggregator, and adds models to the aggregator.

```
1  def __dynamic_aggregator(self, aggregated_models_weights,
       malicious_nodes):
2      """
3      Args:
4          aggregated_models_weights (dict): A dictionary
               containing the aggregated models and their
               corresponding weights.
5          malicious_nodes (list): A list of nodes that are
               considered malicious.
6      """
7      # Select a random aggregation function
8      self.target_aggregation = self.
           __randomly_select_aggregation_function()
9      # Change the aggregator to the target aggregation
           function
10     self.aggregator = self.target_aggregation
11     self.aggregator.set_nodes_to_aggregate(self.__train_set)
12     # Set the round of the aggregator to the current round
13     self.aggregator.set_round(self.round)
14     # Add the models to the aggregator
15     for subnodes in aggregated_models_weights.keys():
16         sublist = subnodes.split()
17         (submodel, weights) = aggregated_models_weights[
               subnodes]
18         # Add the model to the aggregator if the node is not
               in the list of malicious nodes
19         for node in sublist:
20             if node not in malicious_nodes:
21                 self.aggregator.add_model(
22                     submodel, [node], weights, source=self.
                           get_name(), round=self.round
23                 )
```

Listing 5.5: __dynamic_aggregator()

In `__train_step(self)`, the proactive mode and the first case in reactive mode is implemented: In proactive mode, at the start of each training round, if the node itself is not malicious, then the node randomly selects a new aggregation function from the pool to use for this round. In reactive mode, if malicious nodes are detected and the node itself is not malicious, it will broadcast the list of detected malicious nodes to other nodes in the

network, then changes to a new aggregation function, and excludes the malicious from the aggregation function.

```
def __train_step(self):
  ...
  # dynamic aggregation function
  if self.with_reputation and not self.__is_malicious:
    malicious_nodes = []
    # Proactive dynamic aggregation function
    if self.round > 0:
      if (self.is_dynamic_aggregation) and (self.
         dynamic_aggregation_mode == "Proactive"):
        self.__dynamic_aggregator(self.aggregator.
           get_aggregated_models_weights(), malicious_nodes)
    # Reactive dynamic aggregation function
    if self.round > 2:
      if (self.is_dynamic_aggregation) and (self.
         dynamic_aggregation_mode == "Reactive"):
        # Calculate the malicious nodes and reputation score
        malicious_nodes, reputation = self.
           reputation_calculation(self.aggregator.
           get_aggregated_models_weights())

        if len(malicious_nodes) > 0:
          # Send reputation message to other nodes
          self.send_reputation(malicious_nodes)

          # Call the dynamic aggregator function, change the
             aggregator to the target_aggregation
          self.__dynamic_aggregator(self.aggregator.
             get_aggregated_models_weights(), malicious_nodes
             )
```

Listing 5.6: __train_step()

`__reputation_callback()` defines the second case in the reactive mode: if a non-malicious node receives messages about malicious nodes in the network, it will randomly select a new aggregation function and exclude the malicious nodes from the aggregation process.

```
def __reputation_callback(self, msg):
# Receive malicious nodes information from neighbors (
   broadcast REPUTATION message)
malicious_nodes = msg.args  # List of malicious nodes
if self.with_reputation:
    if len(malicious_nodes) > 0 and not self.__is_malicious
       and self.get_name() not in malicious_nodes:
        # Check if dynamic aggregation with Reactive mode is
           enabled
```

```
7           if (self.is_dynamic_aggregation) and (self.
               dynamic_aggregation_mode == "Reactive"):
8               # Call the dynamic aggregator function, change
                   the aggregator to the target aggregation
                   function
9               self.__dynamic_aggregator(self.aggregator.
                   get_aggregated_models_weights(),
                   malicious_nodes)
```

Listing 5.7: __reputation_callback()

Chapter 6

Evaluation

This chapter focuses on evaluating Moving Target Defense (MTD) mechanisms that utilize dynamic aggregation functions, including both proactive and reactive approaches. This work will compare their performance against baseline performance on various datasets. The chapter will first outline the experimental setup, followed by the results, and conclude with a discussion of the observations.

6.1 Experiment Setup

The following sections will cover the datasets, their corresponding models, the threat model, and the configuration for implementing MTD in Fedstellar.

6.1.1 Datasets and Models

This work selected three datasets with IID (Independent and Identically Distributed) characteristics and corresponding deep learning models to evaluate the MTD-based defense mechanism implemented within Fedstella.

- MNIST[80] is a widely used benchmark dataset for neural networks, containing 60,000 training images and 10,000 test images of handwritten digits (0-9), each represented as a 28x28 grayscale image. A Multilayer Perceptron (MLP) with a specific architecture is used to learn this dataset. The MLP consists of an input layer with 28*28 neurons feeding into a 256-neuron linear layer, a hidden layer with 256 ReLU activated neurons feeding into a 128-neuron linear layer, and an output layer with 10 softmax activated neurons for classification [81], [82]. The model is trained using the Adam optimizer [83] with a learning rate of 1e-3 and cross-entropy loss, with 3 epochs per federated round.

Figure 6.1: Model Poisoning Configuration in Fedstellar

- FashionMNIST (FMNIST)[84] is a dataset designed as a more challenging replacement for the classic MNIST dataset. It consists of 60,000 training and 10,000 test images of fashion items across 10 classes (e.g., t-shirt, trouser, dress). Like MNIST, each image is a 28x28 grayscale image. The increased complexity of FMNIST provides a more robust benchmark for evaluating image classification algorithms. The same MLP architecture used for MNIST is also applied to the FMNIST dataset, with a training duration of 3 epochs per round.
- CIFAR10 [85] is another popular benchmark in computer vision. It comprises 60,000 color images (32x32 pixels) across 10 classes (e.g., airplane, bird, cat, truck), with 50,000 training images and 10,000 test images. The complexity of color images and the variety of object classes make CIFAR10 a more challenging dataset for image classification compared to MNIST or FMNIST. Convolutional Neural Networks (CNNs) [86] are a specialized type of neural network designed to excel in image-related tasks. CNNs use convolutional layers to extract and learn features from image data. These layers use kernels (filters) that slide over the image to detect patterns like edges, textures, and shapes. CNNs typically include pooling layers to reduce computation and introduce some translation invariance, along with fully connected layers for final classification.

6.1.2 Model Poisoning Configuration

This work selects Model Poisoning attacks to evaluate the performance of newly proposed MTD defense mechanism. In this attack scenario, a malicious participant intentionally corrupts the integrity of the model by injecting a noise into the local model updates. To characterize the model poisoning attacks, the following key parameters are introduced, just like the configurations of Fedstellar in Figure6.1:

- Poisoned Node Ratio (PNR): The percentage of malicious participants in the federated learning network
- Poisoned Sample Ratio (PSR): The percentage of data samples or labels that an attacker modifies
- Noise Ratio (NR): How much noise an attacker injects into the local data or model

The PNR was set to three different levels: 10%, 50%, and 80%, representing low, medium, and high poisoning environments, respectively. This allows for the evaluation of the

defense mechanisms under varying degrees of malicious participation in the network. For model poisoning attacks, the PSR is not applicable since the attacker directly manipulates the local model parameters instead of tampering with the training data or labels. A salt NR of 30% is chosen for the model poisoning attacks. This level of salt noise is sufficient to degrade the model's performance. The malicious nodes were randomly selected from the pool of participants. Once selected, these nodes launched model poisoning attacks by injecting 30% salt noise into their local model updates before sending them to the central server for aggregation.

In the context of defending against poisoning attacks in federated learning, both proactive and reactive dynamic aggregation techniques are employed.

6.1.3 Fedstellar Configuration

The following paragraphs explain how Fedstellar is configured internally. The experiment utilized a small network of 10 nodes. The topology was fully connected, meaning each node was directly connected to all other nodes. In other words, each node had 9 neighbors, as shown in Figure 6.2. The neighbor assignments remained fixed throughout the experiment. The malicious nodes were randomly placed, with a maximum of 80% of the nodes potentially being malicious. Importantly, all nodes in the network acted as aggregators, participating in both the aggregation process and model training. There were no dedicated proxy nodes.

This work distributed data equally among all nodes using an IID (Independent and Identically Distributed) approach. For MNIST and FMNIST, each node received 6,000 training samples and 1,000 test samples. With CIFAR-10, each node had 5,000 training samples and 1,000 test samples. A 10% validation set was created from the training data for all experiments. Regarding the aggregation algorithm, FedAvg is set as the default aggregation algorithm for all nodes. Benign nodes employ different algorithms later based on whether the reputation system is enabled and its configuration (proactive or reactive).

For all experiments, the federated learning (FL) process was configured to run for 10 rounds. Note that Fedstellar performs a model diffusion at round 0, resulting in a total of 11 rounds. However, local training only takes place during the final 10 rounds. This duration allows sufficient time for attackers to potentially poison the models of benign participants. A complete list of Fedstellar-specific configuration parameters is provided in Table.

All experiments were conducted using Fedstellar's Docker simulation capabilities. The experiments were run on a remote Ubuntu server. For the MNIST, FMNIST, and CIFAR10 datasets, a GPU environment was utilized to accelerate the training process. GPU environment: (2) NVIDIA NVIDIA Tesla T4, CUDA Version: 12.3, Driver Version: 545.23.08, 16 GB Memory, AMD EPYC 7702 64-Core Processor CPU @ 2.0GHz, 16 Sockets, 3-level cache hierarchy, 61 GB RAM, OS: Ubuntu 20.04.6 LTS.

To summarize the experiment setting, for each dataset(MNIST, FMNIST, CIFAR-10), conduct experiments with varying PNR of model poisoning: 0%, 10%, 50%, and 80% and

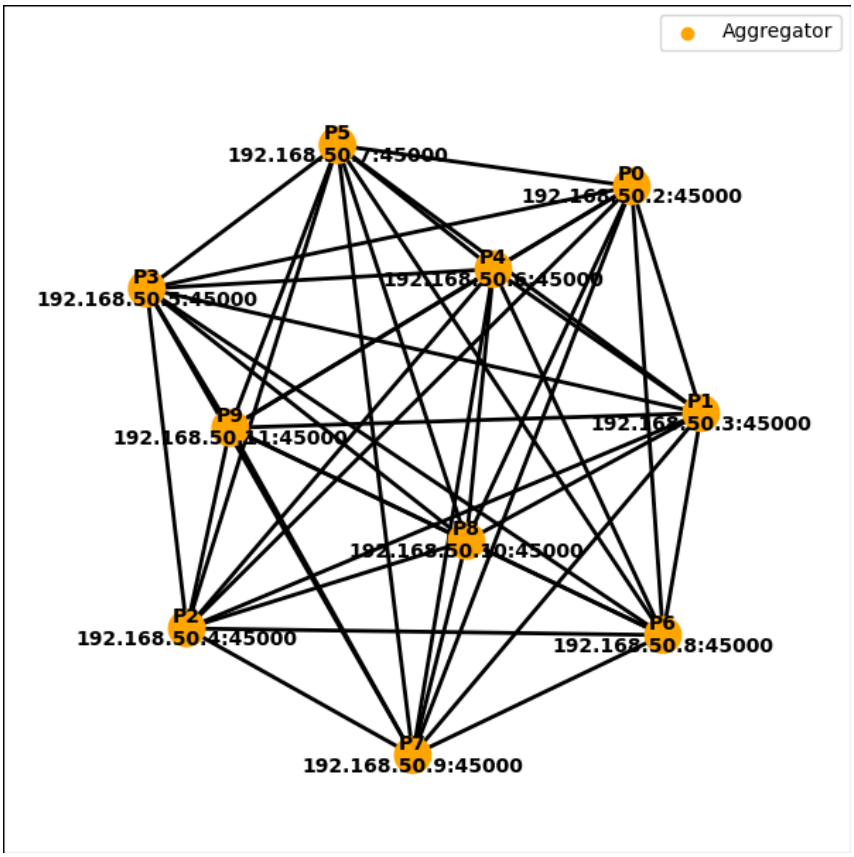


Figure 6.2: A fully connected network of 10 nodes was used as the Fedstellar network configuration for all experiments

a fixed salt NR of 30%. In all poisoning scenarios, three defense configurations were tested: No defense; Proactive MTD defense only; Reactive MTD defense only; The no-defense serves as a baseline. Experimental configurations are detailed in Table 6.1

6.1.4 Evaluation metrics

Two main categories of evaluation metrics are employed to assess the performance and efficiency of the system.

- **Resource-related Metrics:**

  These metrics focus on measuring the resource consumption and utilization during the federated learning process. They provide insights into the computational and communication overhead associated with the decentralized learning approach. The resource-related metrics include:

  - Communication between Nodes (in bytes): This metric quantifies the amount of data exchanged between the nodes participating in the federated learning process. It measures the volume of information transmitted over the network, which can impact the overall communication efficiency and bandwidth requirements.
  - CPU Usage (in percent): This metric monitors the utilization of the Central Processing Unit (CPU) resources on each node during the learning process. It indicates the computational intensity and the percentage of CPU capacity being used by the federated learning algorithms.
  - Disk Usage (in percent): This metric tracks the utilization of disk storage on each node. It measures the percentage of disk space being consumed by the federated learning system, including data storage, intermediate results, and model checkpoints.
  - RAM Usage (in percent): This metric gauges the utilization of Random Access Memory (RAM) on each node during the learning process. It reflects the memory requirements and the percentage of available RAM being used by the federated learning algorithms.

- **Performance-related Metrics:** These metrics assess the effectiveness and quality of the federated learning models in terms of their predictive capabilities and overall performance. They provide a quantitative measure of how well the models are learning and generalizing from the distributed data. The performance-related metrics include:

  - Accuracy: This metric measures the proportion of correct predictions made by the federated learning model compared to the total number of predictions. It indicates the overall correctness and effectiveness of the model in making accurate classifications or predictions.

Table 6.1: Experimental setup for evaluating the impact of model poisoning attacks and dynamic aggregation on Fedstellar. MP stands for Model Poisoning

id	Nodes	Topology	Dataset	Model	Attack	PNR	PSR	NR	MTD
1	10	Fully	MNIST	MLP	-	-	-	-	-
2	10	Fully	MNIST	MLP	MP	10	-	30	-
3	10	Fully	MNIST	MLP	MP	50	-	30	-
4	10	Fully	MNIST	MLP	MP	80	-	30	-
5	10	Fully	MNIST	MLP	-	-	-	-	Proactive
6	10	Fully	MNIST	MLP	MP	10	-	30	Proactive
7	10	Fully	MNIST	MLP	MP	50	-	30	Proactive
8	10	Fully	MNIST	MLP	MP	80	-	30	Proactive
9	10	Fully	MNIST	MLP	-	-	-	-	Reactive
10	10	Fully	MNIST	MLP	MP	10	-	30	Reactive
11	10	Fully	MNIST	MLP	MP	50	-	30	Reactive
12	10	Fully	MNIST	MLP	MP	80	-	30	Reactive
13	10	Fully	FashionMNIST	MLP	-	-	-	-	-
14	10	Fully	FashionMNIST	MLP	MP	10	-	30	-
15	10	Fully	FashionMNIST	MLP	MP	50	-	30	-
16	10	Fully	FashionMNIST	MLP	MP	80	-	30	-
17	10	Fully	FashionMNIST	MLP	-	-	-	-	Proactive
18	10	Fully	FashionMNIST	MLP	MP	10	-	30	Proactive
19	10	Fully	FashionMNIST	MLP	MP	50	-	30	Proactive
20	10	Fully	FashionMNIST	MLP	MP	80	-	30	Proactive
21	10	Fully	FashionMNIST	MLP	-	-	-	-	Reactive
22	10	Fully	FashionMNIST	MLP	MP	10	-	30	Reactive
23	10	Fully	FashionMNIST	MLP	MP	50	-	30	Reactive
24	10	Fully	FashionMNIST	MLP	MP	80	-	30	Reactive
25	10	Fully	CIFAR10	CNN	-	-	-	-	-
26	10	Fully	CIFAR10	CNN	MP	10	-	30	-
27	10	Fully	CIFAR10	CNN	MP	50	-	30	-
28	10	Fully	CIFAR10	CNN	MP	80	-	30	-
29	10	Fully	CIFAR10	CNN	-	-	-	-	Proactive
30	10	Fully	CIFAR10	CNN	MP	10	-	30	Proactive
31	10	Fully	CIFAR10	CNN	MP	50	-	30	Proactive
32	10	Fully	CIFAR10	CNN	MP	80	-	30	Proactive
33	10	Fully	CIFAR10	CNN	-	-	-	-	Reactive
34	10	Fully	CIFAR10	CNN	MP	10	-	30	Reactive
35	10	Fully	CIFAR10	CNN	MP	50	-	30	Reactive
36	10	Fully	CIFAR10	CNN	MP	80	-	30	Reactive

- Loss: This metric quantifies the discrepancy between the predicted values and the actual values. It represents the error or cost associated with the model's predictions. Lower loss values indicate better model performance and closer alignment with the ground truth.
- Precision: This metric calculates the proportion of true positive predictions among all positive predictions made by the model. It focuses on the model's ability to correctly identify positive instances while minimizing false positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{6.1}$$

- Recall: This metric measures the proportion of true positive predictions among all actual positive instances in the dataset. It assesses the model's ability to correctly identify positive instances while minimizing false negatives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6.2}$$

- F1-Score: This metric combines precision and recall into a single value, providing a balanced measure of the model's performance. It is the harmonic mean of precision and recall and offers a comprehensive evaluation of the model's accuracy, considering both false positives and false negatives.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.3}$$

By employing these two categories of evaluation metrics, Fedstellar can comprehensively assess the resource efficiency and predictive performance of the models. The resource-related metrics help optimize the utilization of computational resources and communication bandwidth, while the performance-related metrics enable the evaluation and comparison of the MTD defense mechanism in terms of their effectiveness and generalization capabilities.

6.2 Results

This section investigates how dynamic aggregation techniques resist model poisoning attacks. First, establishe a performance benchmark by measuring the system's performance under normal conditions (no attacks). Then, analyze the impact of model poisoning attacks on the federated learning system across all datasets (MNIST, FMNIST, CIFAR-10). Finally, introduce proactive and reactive dynamic aggregation defenses, evaluating their effectiveness in mitigating these attacks. By examining how attacks affect each dataset, this work assesses the robustness of the dynamic aggregation techniques

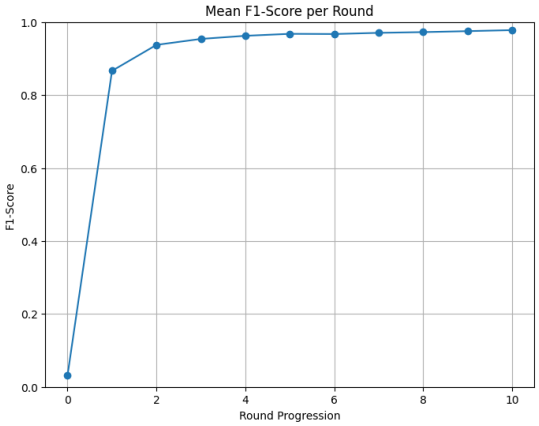
6.2.1 Baseline Performance

A baseline performance measurement enables the evaluation of a defense mechanism's effectiveness under attack conditions. Figure 6.3 establish the baseline performance of federated learning models on these datasets under benign conditions, without any attacks or defenses applied. the baseline performance of federated learning models on three different datasets: MNIST, Fashion-MNIST (FMNIST), and CIFAR-10, over the course of 10 training rounds. The performance is evaluated using two metrics: F1-Score and Loss. This baseline serves as a reference for evaluating the impact of attacks and the effectiveness of defense mechanisms in subsequent experiments.

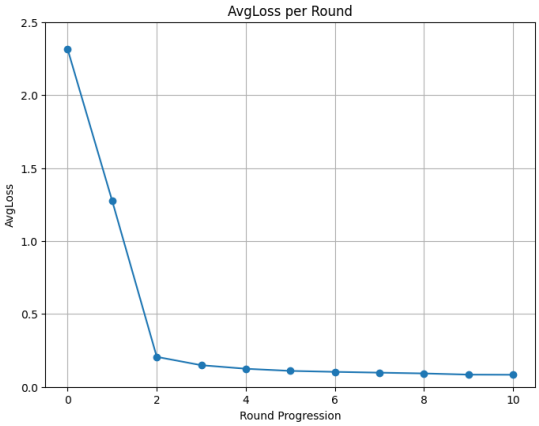
For MNIST dataset, the F1-score starts around 0.2 and increases to around 0.8 at round 1, then steadily increases to around 0.9 by the end of learning. The loss starts around 2.2 and dereases to around 0.6 by round 10, reflecting the reduction in error as the model learns. In Fashion-MNIST, the F1-score and loss follows a similar trend as MNIST. The CIFAR-10 F1-Score starts lower, around 0.1, but increases to around 0.7 by round 10, suggesting a more challenging dataset. The CIFAR-10 Loss starts higher, around 2.5, and decreases to around 1.0 by round 10, indicating a higher error rate compared to the other datasets.

Figure 6.4 show the resources usage during a time period of 10 minutes. It contains three line charts labeled (a) Network Baseline, (b) CPU Baseline, and (c) RAM Baseline. Each chart displays the resource usage over time for datasets: MNIST, FMNIST, and CIFAR10.

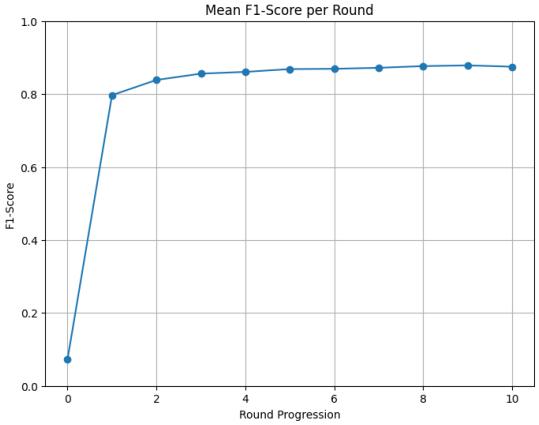
In Network usage, MNIST has the lowest network usage, followed by FMNIST, with CIFAR10 requiring the highest network resources. The network usage increases gradually for all datasets as training progresses, with CIFAR10 showing the steepest increase. In CPU usage, the lines show oscillating CPU usage over time for each dataset, with CIFAR10 again requiring the highest CPU resources on average, representing the iterative training process with varying computational demands in each iteration. Similar to network usage, in RAM usage, the memory requirements increase gradually over time for all datasets, with CIFAR10 again consuming the most RAM. In summary, the more complex CIFAR10 dataset demands significantly more resources across all metrics compared to the simpler MNIST and FMNIST datasets.



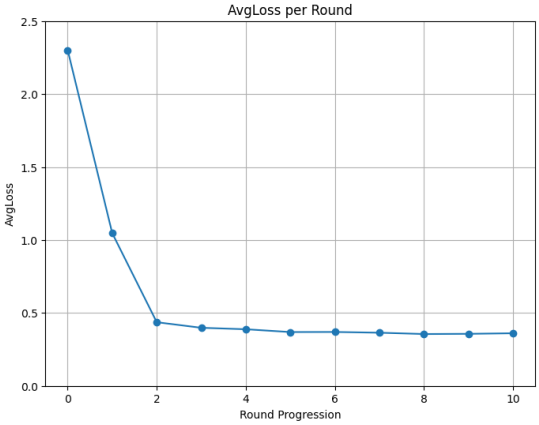
(a) MNIST F1-Score



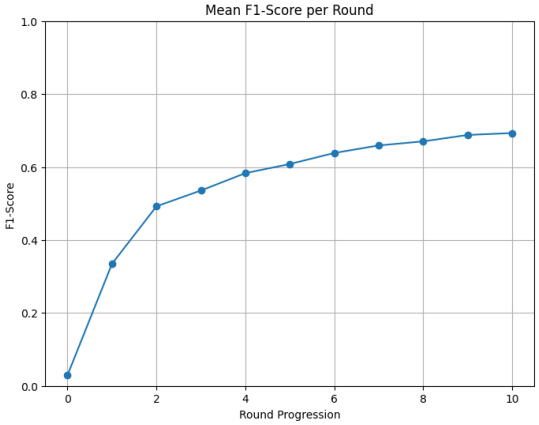
(b) MNIST Loss



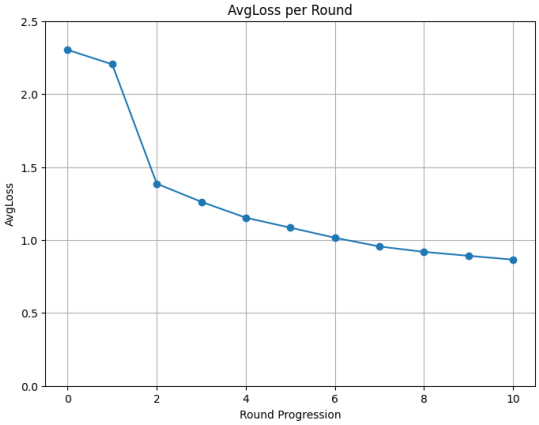
(c) FMNIST F1-Score



(d) FMNIST Loss

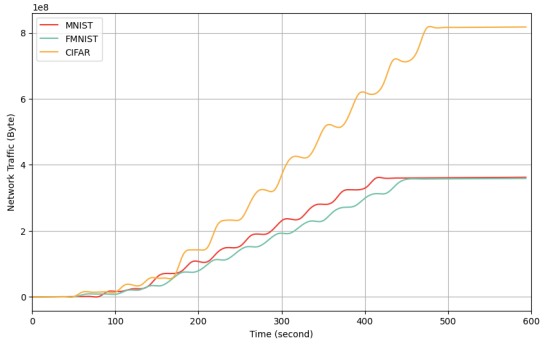


(e) CIFAR10 F1-Score

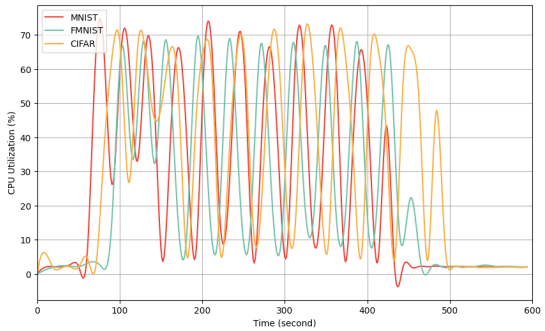


(f) CIFAR10 Loss

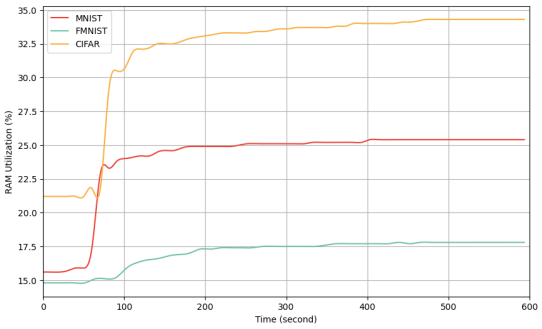
Figure 6.3: Baseline F1-Score and Loss performance on MNIST, FMNIST, and CIFAR10 over 10 rounds.



(a) Network Baseline



(b) CPU Baseline



(c) RAM Baseline

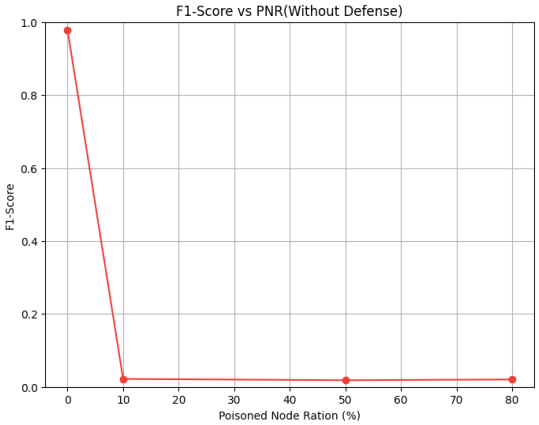
Figure 6.4: Baseline Resources on MNIST, FMNIST, and CIFAR10 over 10 rounds.

6.2.2 Model Poisoning

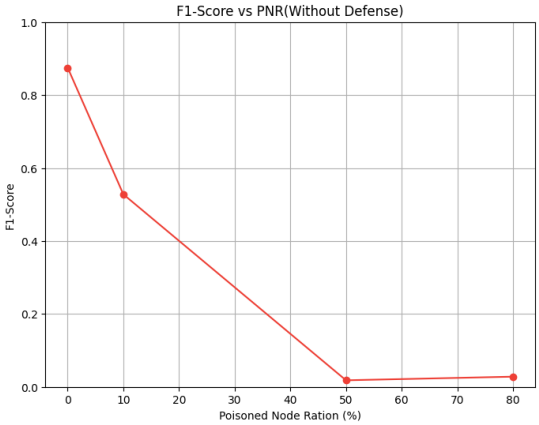
In a model poisoning attack, a malicious participant deliberately corrupts the model's weights with noise during local training, while leaving the training data untouched. This corrupted model is then shared with neighbors during communication rounds. In this work's experiments, a fixed salt Noise Ration(NR) of 30% is selected. The Poisoned Node Ration(PNR) is set to three different levels: 10%, 50% and 80%.

F1-Score

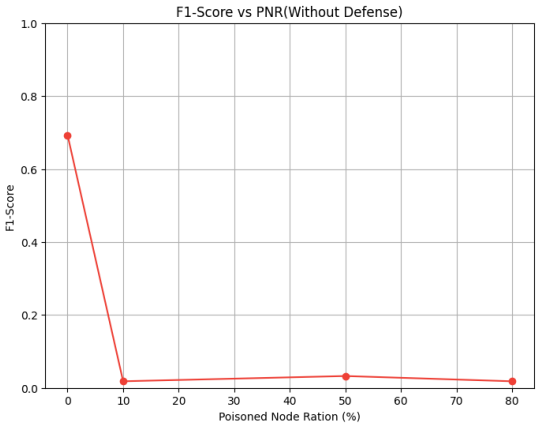
The set of graphs (a), (b), and (c) in Figure6.5 show the F1-Score plotted against the PNR for three different datasets: MNIST, Fashion-MNIST (FMNIST), and CIFAR-10, respectively. In each graph, the x-axis represents the percentage of poisoned nodes in the training data, while the y-axis represents the F1-Score achieved by the model. In all three graphs, the F1-Score decreases as the Poisoned Node Ratio increases, indicating that the presence of malicious nodes degrades the model's performance. The rate of decrease varies across datasets, with CIFAR-10 (c) showing a steeper decline compared to MNIST (a) and Fashion-MNIST (b).



(a) MNIST F1-Score(without defense)



(b) FMNIST F1-Score(without defense)

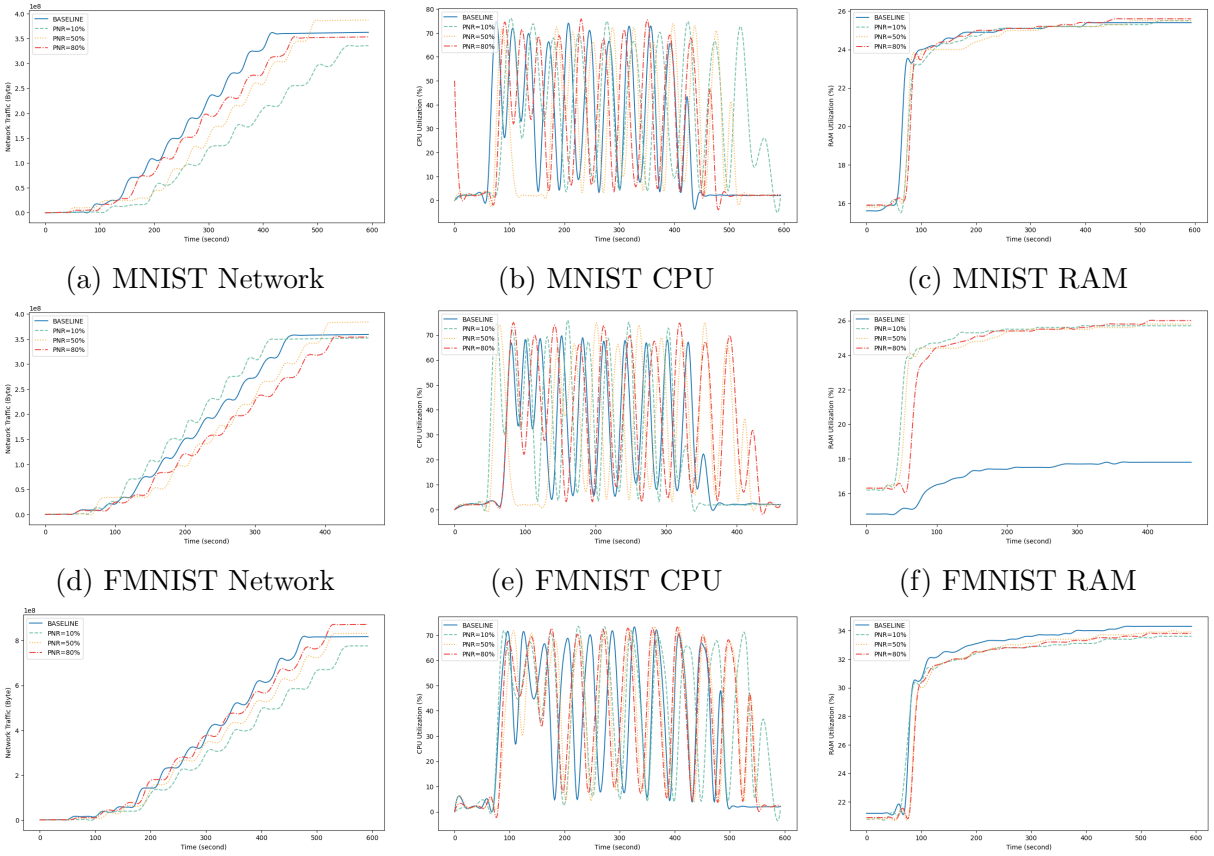


(c) CIFAR10 F1-Score(without defense)

Figure 6.5: F1-score(Without Defense) in three topologies for MNIST (a), FashionMNIST (b), and Cifar10 (c) Datasets with Model Poisoning Attack

Resource

Figure 6.6 shows the network, cpu and ram usage for different datasets under different PNR ratios. There is a drop of network traffic on MNIST dataset. The FMNIST dataset experiences a great increase in RAM usage. But overall, adding model poisoning attacks to training does not significantly increase resource consumption.



(a) MNIST Network (b) MNIST CPU (c) MNIST RAM

(d) FMNIST Network (e) FMNIST CPU (f) FMNIST RAM

(g) CIFAR Network (h) CIFAR CPU (i) CIFAR RAM

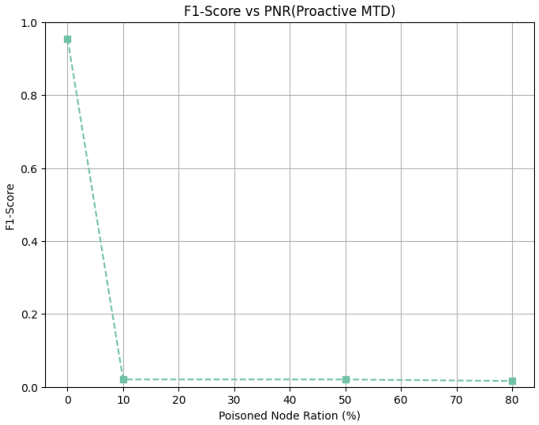
Figure 6.6: Resource usage(without defense) of different datasets under varied PNR.

6.2.3 Proactive MTD

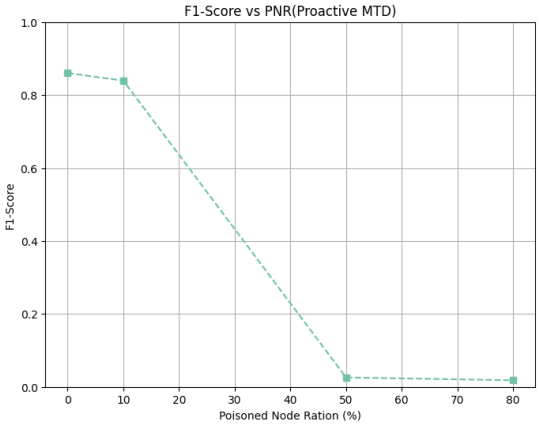
The Moving Target Defense mechanism implemented here is dynamic aggregation. In its proactive mode, non-malicious nodes randomly choose a new aggregation function from the pool at the beginning of each training round.

F1-Score

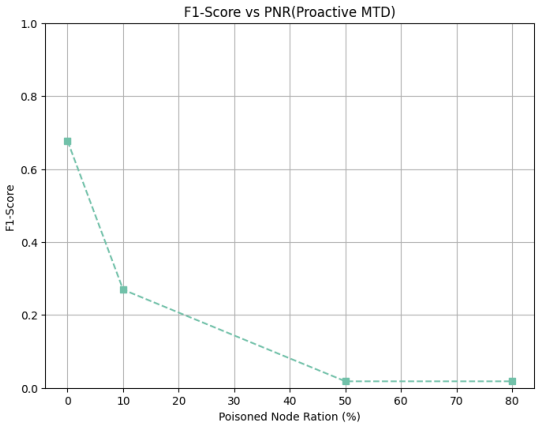
Figure 6.7 demonstrates the impact of increasing Poisoned Node Ratio (PNR) on F1-Score within a proactive MTD setting. Across MNIST, FMNIST, and CIFAR10 datasets, F1-Scores decline as PNR rises. The MNIST dataset experiences a severe F1-Score drop even at 10% PNR, while FMNIST and CIFAR10 exhibit a more gradual decline at the same level. While proactive MTD offers some mitigation in low poisoning environments, its effectiveness diminishes at higher PNRs. This highlights that proactive MTD can lessen the impact of model poisoning attacks but doesn't provide complete immunity.



(a) MNIST F1-Score(proactive)



(b) FMNIST F1-Score(proactive)

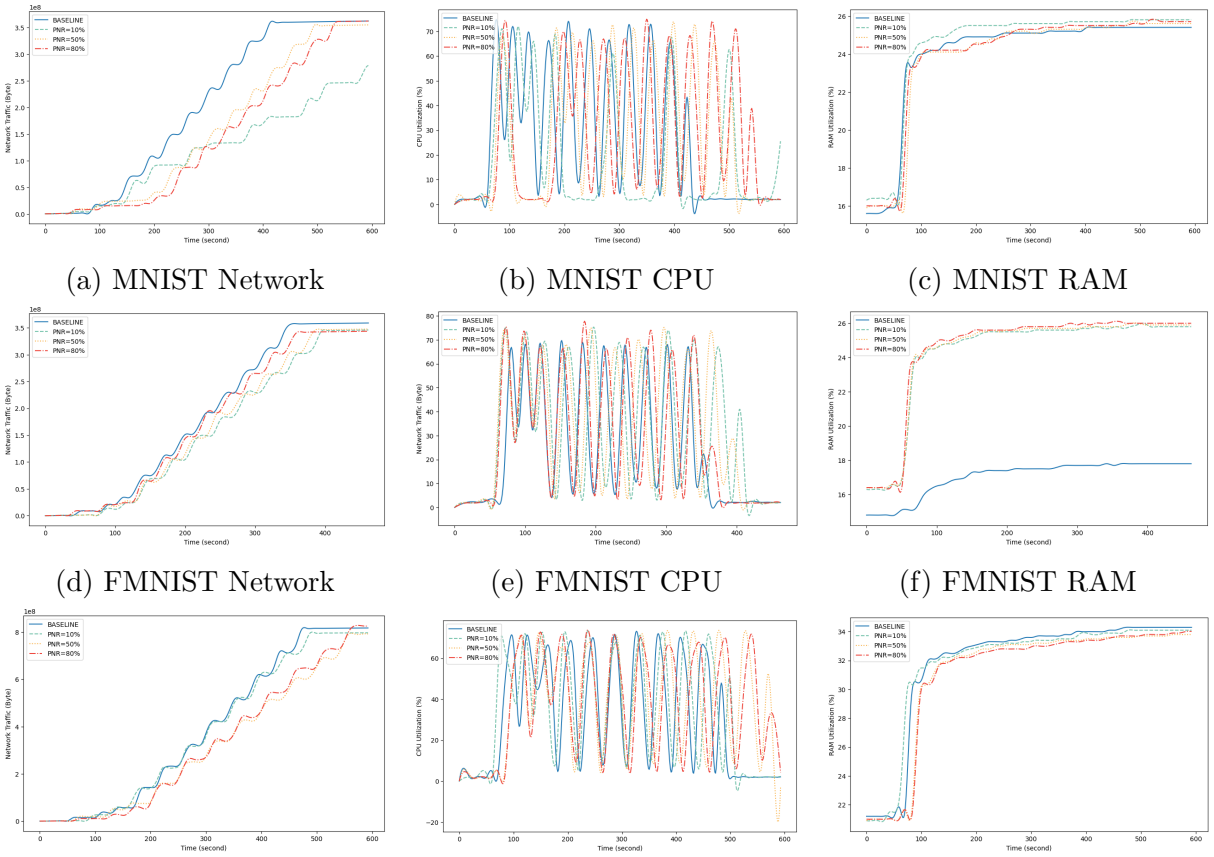


(c) CIFAR10 F1-Score(proactive)

Figure 6.7: F1-score(Proactive MTD) in three topologies for MNIST (a), FashionMNIST (b), and Cifar10 (c) Datasets with Model Poisoning Attack

Resource

Figure 6.8 indicates that proactive MTD does not significantly increase resource consumption (network, CPU, RAM) across different datasets and PNR ratios. The MNIST dataset exhibits lower traffic overhead while the RAM usage on FMNIST dataset increases significantly.



(a) MNIST Network (b) MNIST CPU (c) MNIST RAM

(d) FMNIST Network (e) FMNIST CPU (f) FMNIST RAM

(g) CIFAR Network (h) CIFAR CPU (i) CIFAR RAM

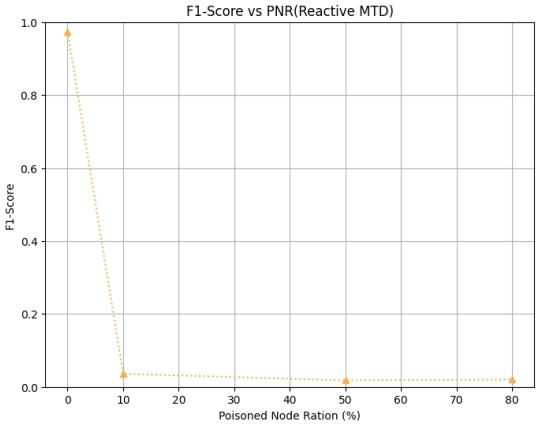
Figure 6.8: Resource usage(proactive) of different datasets under varied PNR.

6.2.4 Reactive MTD

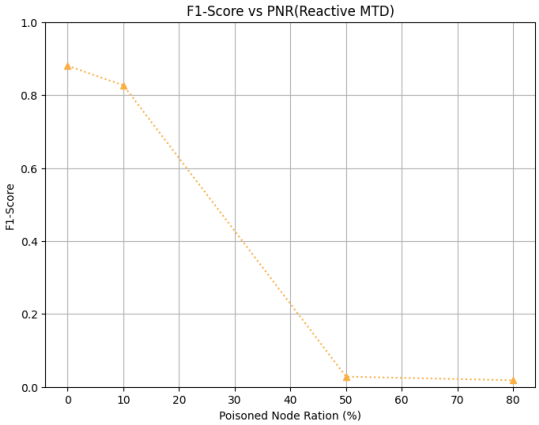
In reactive MTD settings, benign nodes randomly chooses a new aggregation function if malicious nodes are detected.

F1-Score

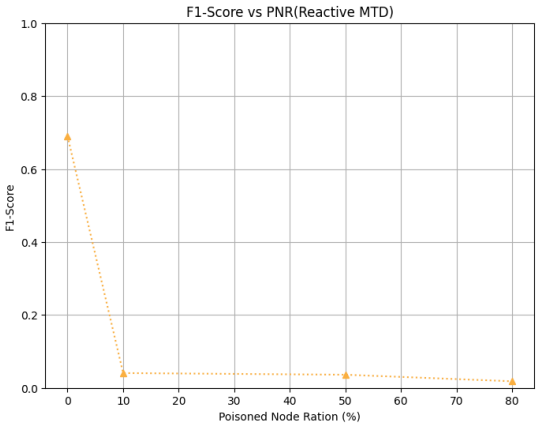
Figure 6.9 demonstrates the impact of increasing Poisoned Node Ratio (PNR) on F1-Score within a reactive MTD setting. Like in the proactive settings, for all three datasets, the F1-Score goes down as the percentage of poisoned data goes up. The graphs also show that the F1-Score is higher for some datasets than others.Reactive MTD demonstrates good performance on the FashionMNIST dataset at 10% PNR, but its effectiveness is low for MNIST and CIFAR-10 datasets.



(a) MNIST F1-Score(reactive)



(b) FMNIST F1-Score(reactive)

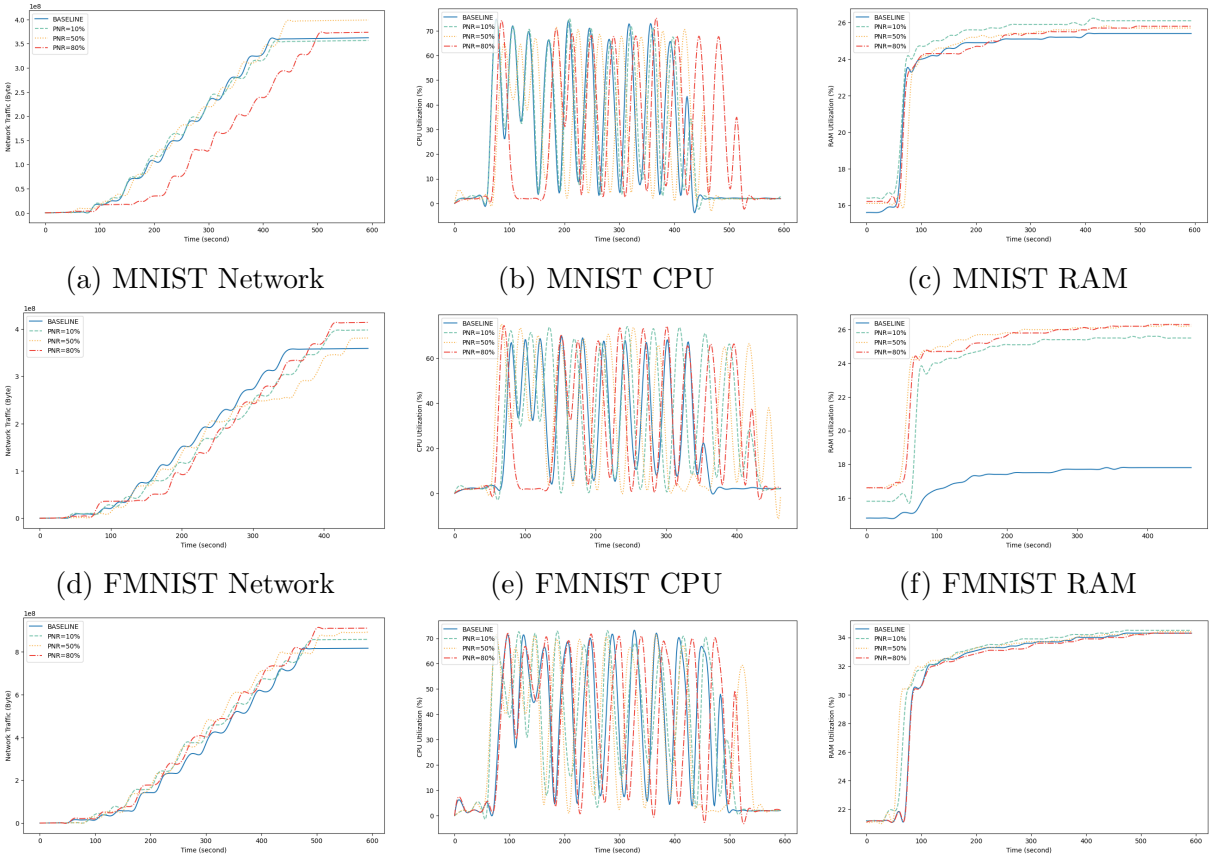


(c) CIFAR10 F1-Score(reactive)

Figure 6.9: F1-score(Reactive MTD) in three topologies for MNIST (a), FashionMNIST (b), and Cifar10 (c) Datasets with Model Poisoning Attack

Resources

Figure 6.8 indicates that reactive MTD does not significantly increase resource consumption (network, CPU, RAM) across different datasets and PNR ratios. Like in the previous charts, FMNIST experiences a higher RAM usage.



(a) MNIST Network (b) MNIST CPU (c) MNIST RAM

(d) FMNIST Network (e) FMNIST CPU (f) FMNIST RAM

(g) CIFAR Network (h) CIFAR CPU (i) CIFAR RAM

Figure 6.10: Resource usage(reactive) of different datasets under varied PNR.

6.3 Discussion

Figure 6.11 compares the F1-Score performance of different defense strategies against model poisoning attacks. There are three lines plotted:

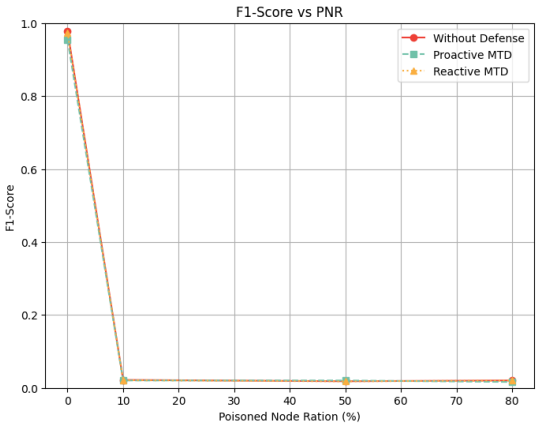
1. Without Defense (red line) - This serves as the baseline. As expected, without any defense, the model's F1-score decreases significantly as the proportion of poisoned nodes (PNR) increases.

2. Proactive MTD (green line) - Proactive Moving Target Defense (MTD) attempts to make attacks harder by continuously changing aspects of the system. It shows improved resilience, maintaining a higher F1-score than 'no defense'. It's particularly effective in cases of low PNR, as evident in the FMNIST graph. However, in higher PNR, the effect of proactive MTD diminishes.

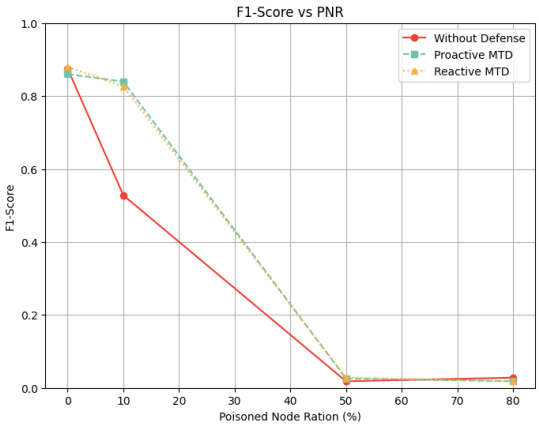
3. Reactive MTD (blue line) - Reactive MTD responds to detected attacks. Its performance varies. In the FMNIST dataset, it shows some improvement, but in other cases, it's less effective compared to proactive MTD.

To summarize, for the MNIST dataset, the MTD mechanisms have minimal impact in mitigating the effects of poisoned data on the F1-score performance. However, for the FashionMNIST and CIFAR10 datasets, the MTD mechanisms are more effective, especially in low poisoning environments with smaller percentages of poisoned training data. Among the different MTD approaches, the proactive MTD strategy generally provides the most robust defense against poisoned node attacks, particularly when the proportion of poisoned data is relatively low. Across most poisoned node ratios (PNRs), the proactive MTD outperforms the other defense mechanisms. The reactive MTD approach can offer some improvement over no defense, but its reliability may vary across different scenarios and poisoning levels. While the implementation of dynamic aggregation techniques can help reduce the impact of model poisoning attacks, it does not provide complete immunity. The effects of the poisoning attacks are still observed, albeit to a lesser degree compared to having no defense mechanism.

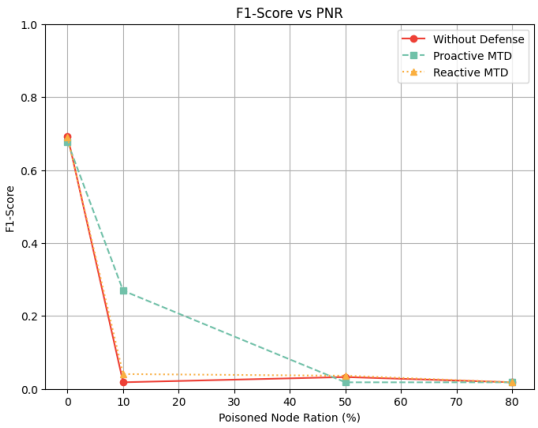
In terms of resources, model poisoning can lead to a decrease in network traffic on MNIST and a significant increase in RAM usage on FashionMNIST. Incorporating proactive and reactive MTD techniques has a negligible additional impact on network, CPU, and RAM usage.



(a) MNIST F1-Score



(b) FMNIST F1-Score



(c) CIFAR10 F1-Score

Figure 6.11: F1-score in three topologies for MNIST (a), FashionMNIST (b), and Cifar10 (c) Datasets with Model Poisoning Attack

Chapter 7

Summary and Conclusions

7.1 Conclusion

The purpose of this project is to develop and implement a mitigation strategy utilizing MTD techniques to strengthen the security and resilience of Decentralized Federated Learning systems against poisoning attacks. By dynamically changing the aggregation functions used by participants, this approach seeks to strengthen system resilience against model poisoning, particularly in low-poisoning scenarios across selected datasets, while minimizing additional resource consumption. Current research on the MTD method for DFL is relatively scarce and lacks a systematic summary and analysis of existing work. The work fills a gap in current research by exploration and development of MTD methods tailored for the unique challenges of DFL systems.

The theoretical background served as an introduction to on Federated Learning, poisoning attacks, and Moving Target Defense. This establishes the foundation for understanding the security challenges in DFL and the potential of MTD techniques. With the background knowledge this thesis develops a comprehensive framework for understanding and categorizing MTD techniques in DFL. The framework explores various implementation perspectives, including shuffling, diversity, redundancy, and hybrid techniques, as well as possible attack surfaces in DFL. The usage, advantages, and disadvantages of each technique are analyzed and discussed. With the framework, this thesis proposes a novel MTD mechanism called Dynamic Aggregation Functions, also known as Randomized Aggregation Functions, to mitigate poisoning attacks in DFL. Dynamic Aggregation empowers nodes to proactively or reactively switch aggregation functions during the federated learning process. This introduces flexibility and unpredictability, improving the robustness, adaptability, and overall security of the system against malicious behavior.

The design of the Dynamic Aggregation Functions MTD technique was adapted and implemented on Fedstellar, an open-source platform for training decentralized federated learning models across diverse physical and virtualized devices. Extensive evaluations were then conducted on the MNIST, FashionMNIST (FMNIST), and CIFAR10 datasets to assess the effectiveness of the proposed Dynamic Aggregation Functions against model poisoning attacks under various configurations. The experiments results show that this

MTD technique demonstrates effectiveness in mitigating the impact of model poisoning attacks, particularly in low poisoning environments and when using the proactive approach. However, the technique's performance varied across datasets and did not provide complete protection against attacks. The incorporation of MTD had minimal overhead in terms of resource consumption.

7.2 Limitations and Future Work

As for now, while the proposed dynamic aggregation functions demonstrates effectiveness in mitigating the impact of model poisoning attacks in DFL,it has several limits. First, the performance of the technique varies across datasets and does not provide complete protection against attacks, especially in high poisoning scenarios. Second, this work focused only on model poisoning attacks, while other types of poisoning attacks, such as data poisoning(targeted/untargeted label flipping, sample poisoning), backdoor attacks, were not investigated. Further experiments should be conducted to access the effectiveness of the proposed MTD techniques against a broader range of poisoning attacks. Further, the experiments only analyzed datasets with Independent and Identically Distributed(IID) settings. Using an IID setting ensures that each node has access to data samples that are representative of the overall data distribution. This promotes fairness and prevents any single node from having an outsized influence due to having substantially different data compared to others. As future findings improve the convergence of DFL in non-IID scenarios, the defense techniques should also be evaluated im more heterogeneous environments. Moreover, the experiments were conducted in a fully connected network topology with ten nodes, representing a worst-case scenario for DFL security. Future research should explore the performance of the proposed defense technique in other network topologies such as Ring, Star, Random. Lastly, while the work proposed dynamic aggregation functions as an MTD technique, it did not extensively explore other MTD design dimensions, such as shuffling, diversity, or hybrid approaches. Investigating the potential of combining different MTD techniques could lead to more robust and adaptive defense strategies.

Bibliography

[1] A. Panda, S. Mahloujifar, A. N. Bhagoji, S. Chakraborty, and P. Mittal, "Sparsefed: Mitigating model poisoning attacks in federated learning with sparsification", in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2022, pp. 7587–7624.

[2] E. T. Martínez Beltrán, P. M. Sánchez Sánchez, S. López Bernal, *et al.*, "Mitigating communications threats in decentralized federated learning through moving target defense", *Wireless Networks*, pp. 1–15, 2024.

[3] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges", *IEEE Communications Surveys & Tutorials*, 2023.

[4] A. K. Nair, E. D. Raj, and J. Sahoo, "A robust analysis of adversarial attacks on federated learning environments", *Computer Standards & Interfaces*, vol. 86, p. 103 723, 2023.

[5] A. Uprety and D. B. Rawat, "Mitigating poisoning attack in federated learning", in *2021 IEEE symposium series on computational intelligence (SSCI)*, IEEE, 2021, pp. 01–07.

[6] Z. Tian, L. Cui, J. Liang, and S. Yu, "A comprehensive survey on poisoning attacks and countermeasures in machine learning", *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–35, 2022.

[7] N. Truong, K. Sun, S. Wang, F. Guitton, and Y. Guo, "Privacy preservation in federated learning: An insightful survey from the gdpr perspective", *Computers & Security*, vol. 110, p. 102 402, 2021.

[8] Z. Zhou, C. Xu, M. Wang, T. Ma, and S. Yu, "Augmented dual-shuffle-based moving target defense to ensure cia-triad in federated learning", in *2021 IEEE Global Communications Conference (GLOBECOM)*, IEEE, 2021, pp. 01–06.

[9] M. Asad, A. Moustafa, and T. Ito, "Federated learning versus classical machine learning: A convergence comparison", *arXiv preprint arXiv:2107.10976*, 2021.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data", in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.

[11] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning", *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[12] M. Alazab, S. P. RM, M. Parimala, P. K. R. Maddikunta, T. R. Gadekallu, and Q.-V. Pham, "Federated learning for cybersecurity: Concepts, challenges, and future directions", *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3501–3509, 2021.

[13] M. Moshawrab, M. Adda, A. Bouzouane, H. Ibrahim, and A. Raad, "Reviewing federated learning aggregation algorithms; strategies, contributions, limitations and future perspectives", *Electronics*, vol. 12, no. 10, p. 2287, 2023.

[14] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions", *IEEE signal processing magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[15] I. Research, *What is federated learning?*, Accessed: 2023-03-19, 2023. [Online]. Available: `https://research.ibm.com/blog/what-is-federated-learning`.

[16] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints", *IEEE transactions on neural networks and learning systems*, vol. 32, no. 8, pp. 3710–3722, 2020.

[17] L. Yuan, L. Sun, P. S. Yu, and Z. Wang, "Decentralized federated learning: A survey and perspective", *arXiv preprint arXiv:2306.01603*, 2023.

[18] Baeldung, "Federated learning", *baeldung.com*, 2023. [Online]. Available: `https://www.baeldung.com/cs/federated-learning`.

[19] S. Kamei and S. Taghipour, "A comparison study of centralized and decentralized federated learning approaches utilizing the transformer architecture for estimating remaining useful life", *Reliability Engineering & System Safety*, vol. 233, p. 109 130, 2023.

[20] S. Kalra, J. Wen, J. C. Cresswell, M. Volkovs, and H. R. Tizhoosh, "Decentralized federated learning through proxy model sharing", *Nature communications*, vol. 14, no. 1, p. 2899, 2023.

[21] P. Kairouz, B. McMahan, and V. Smith, "Federated learning and analytics: Industry meets academia", *NeurIPS 2020 tutorial*, 2020.

[22] Y. Sun, L. Shen, and D. Tao, "Which mode is better for federated learning? centralized or decentralized", *arXiv preprint arXiv:2310.03461*, 2023.

[23] N. Bouacida and P. Mohapatra, "Vulnerabilities in federated learning", *IEEE Access*, vol. 9, pp. 63 229–63 249, 2021.

[24] V. Mothukuri, R. M. Parizi, S. Pouriyeh, Y. Huang, A. Dehghantanha, and G. Srivastava, "A survey on security and privacy of federated learning", *Future Generation Computer Systems*, vol. 115, pp. 619–640, 2021.

[25] J. Liu, J. Huang, Y. Zhou, *et al.*, "From distributed machine learning to federated learning: A survey", *Knowledge and Information Systems*, vol. 64, no. 4, pp. 885–917, 2022.

[26] P. M. Mammen, "Federated learning: Opportunities and challenges", *arXiv preprint arXiv:2101.05428*, 2021.

[27] C. Fu, X. Zhang, S. Ji, *et al.*, "Label inference attacks against vertical federated learning", in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 1397–1414.

[28] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting gradients-how easy is it to break privacy in federated learning?", *Advances in Neural Information Processing Systems*, vol. 33, pp. 16 937–16 947, 2020.

[29] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients", *Advances in neural information processing systems*, vol. 32, 2019.

[30] G. Xia, J. Chen, C. Yu, and J. Ma, "Poisoning attacks in federated learning: A survey", *IEEE Access*, vol. 11, pp. 10 708–10 722, 2023.

[31] C. Feng, Y. Zhang, and X. Wang, "Voyager: Mtd-based aggregation protocol for mitigating poisoning attacks on dfl", *arXiv preprint arXiv:2310.08739*, 2024.

[32] A. Shafahi, W. R. Huang, M. Najibi, *et al.*, "Poison frogs! targeted clean-label poisoning attacks on neural networks", in *Advances in Neural Information Processing Systems*, 2018, pp. 6103–6113.

[33] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning", in *IEEE Symposium on Security and Privacy*, IEEE, 2018, pp. 19–35.

[34] C. Xie, K. Huang, P.-Y. Chen, and B. Li, "Dba: Distributed backdoor attacks against federated learning", in *International Conference on Learning Representations*, 2020.

[35] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning", in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 2938–2948.

[36] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain", *arXiv preprint arXiv:1708.06733*, 2017.

[37] E. Bagdasaryan and V. Shmatikov, "Blind backdoors in deep learning models", *arXiv preprint arXiv:2005.03823*, 2020.

[38] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning", in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 1354–1371.

[39] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning", *arXiv preprint arXiv:1712.05526*, 2017.

[40] B. Biggio, I. Corona, D. Maiorca, *et al.*, "Evasion attacks against machine learning at test time", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2013, pp. 387–402.

[41] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?", in *International Conference on Machine Learning*, PMLR, 2015, pp. 1689–1698.

[42] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners", in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015, pp. 2871–2877.

[43] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, "Analyzing federated learning through an adversarial lens", in *International Conference on Machine Learning*, PMLR, 2019, pp. 634–643.

[44] C. Fung, C. Yoon, and I. Beschastnikh, "Mitigating sybils in federated learning poisoning. arxiv 2018", *arXiv preprint arXiv:1808.04866*,

[45] R. Guerraoui, S. Rouault, *et al.*, "The hidden vulnerability of distributed learning in byzantium", in *International Conference on Machine Learning*, PMLR, 2018, pp. 3521–3530.

[46] L. Muñoz-González, B. Biggio, A. Demontis, *et al.*, "Towards poisoning of deep learning algorithms with back-gradient optimization", in *Proceedings of the 10th ACM workshop on artificial intelligence and security*, 2017, pp. 27–38.

[47] M. Fang, X. Cao, J. Jia, and N. Gong, "Local model poisoning attacks to {byzantine-robust} federated learning", in *29th USENIX security symposium (USENIX Security 20)*, 2020, pp. 1605–1622.

[48] J.-H. Cho, D. P. Sharma, H. Alavizadeh, *et al.*, "Toward proactive, adaptive defense: A survey on moving target defense", *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 709–745, 2020.

[49] ". of Homeland Security (.gov)", *"moving target defense (mtd)"*, Accessed: 2023-03-19, 2023. [Online]. Available: `https://www.dhs.gov/science-and-technology/csd-mtd`.

[50] R. Sun, Y. Zhu, J. Fei, and X. Chen, "A survey on moving target defense: Intelligently affordable, optimized and self-adaptive", *Applied Sciences*, vol. 13, no. 9, p. 5367, 2023.

[51] C. Lei, H.-Q. Zhang, J.-L. Tan, Y.-C. Zhang, and X.-H. Liu, "Moving target defense techniques: A survey", *Security and Communication Networks*, vol. 2018, 2018.

[52] J. Zheng and A. S. Namin, "A survey on the moving target defense strategies: An architectural perspective", *Journal of Computer Science and Technology*, vol. 34, no. 1, pp. 207–233, 2019.

[53] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense", *Proceedings of the First ACM Workshop on Moving Target Defense*, pp. 31–40, 2014.

[54] S. Sengupta, A. Chowdhary, A. Sabur, A. Alshamrani, D. Huang, and S. Kambhampati, "A survey of moving target defenses for network security", *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1909–1941, 2020.

[55] L. Bilge and T. Dumitraş, "Before we knew it: An empirical study of zero-day attacks in the real world", in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 833–844.

[56] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, *Moving target defense: creating asymmetric uncertainty for cyber threats*. Springer Science & Business Media, 2011, vol. 54.

[57] M. Torquato and M. Vieira, "Moving target defense techniques: A survey", *Computers & Security*, vol. 88, p. 101 647, 2019.

[58] S. Vikram, C. Yang, and G. Gu, "Nomad: Towards non-intrusive moving-target defense against web bots", in *2013 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2013, pp. 55–63.

[59] Q. Jia, K. Sun, and A. Stavrou, "Motag: Moving target defense against internet denial of service attacks", in *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, IEEE, 2013, pp. 1–9.

[60] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking", in *Proceedings of the first workshop on Hot topics in software defined networks*, 2012, pp. 127–132.

[61] S. Debroy, P. Calyam, M. Nguyen, A. Stage, and V. Georgiev, "Frequency-minimal moving target defense using software-defined networking", in *2016 international conference on computing, networking and communications (ICNC)*, IEEE, 2016, pp. 1–6.

[62] W. Jin, Y. Yao, S. Han, *et al.*, "Fedml-he: An efficient homomorphic-encryption-based privacy-preserving federated learning system", *arXiv preprint arXiv:2303.10837*, 2023.

[63] K. Wei, J. Li, M. Ding, *et al.*, "Federated learning with differential privacy: Algorithms and performance analysis", *IEEE transactions on information forensics and security*, vol. 15, pp. 3454–3469, 2020.

[64] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent", *Advances in neural information processing systems*, vol. 30, 2017.

[65] Z. Zhang, Q. Su, and X. Sun, "Dim-krum: Backdoor-resistant federated learning for nlp with dimension-wise krum-based aggregation", *arXiv preprint arXiv:2210.06894*, 2022.

[66] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates", in *International Conference on Machine Learning*, Pmlr, 2018, pp. 5650–5659.

[67] Y. Qu, L. Gao, T. H. Luan, *et al.*, "Decentralized privacy using blockchain-enabled federated learning in fog computing", *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5171–5183, 2020.

[68] M. R. Behera, S. Upadhyay, S. Shetty, S. Priyadarshini, P. Patel, and K. F. Lee, "Fedsyn: Synthetic data generation using federated learning", *arXiv preprint arXiv:2203.05931*, 2022.

[69] A. Roy, A. Chhabra, C. A. Kamhoua, and P. Mohapatra, "A moving target defense against adversarial machine learning", in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 383–388.

[70] A. Amich and B. Eshete, "Morphence: Moving target defense against adversarial examples", in *Annual Computer Security Applications Conference*, 2021, pp. 61–75.

[71] Y. Qian, Y. Guo, Q. Shao, *et al.*, "Ei-mtd: Moving target defense for edge intelligence against adversarial attacks", *ACM Transactions on Privacy and Security*, vol. 25, no. 3, pp. 1–24, 2022.

[72] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence", *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[73] Y. Qiu, J. Wu, S. Mumtaz, J. Li, A. Al-Dulaimi, and J. J. Rodrigues, "Mt-mtd: Muti-training based moving target defense trojaning attack in edged-ai network", in *ICC 2021-IEEE International Conference on Communications*, IEEE, 2021, pp. 1–6.

[74] S. Sengupta, T. Chakraborti, and S. Kambhampati, "Mtdeep: Boosting the security of deep neural nets against adversarial attacks with moving target defense", in *Decision and Game Theory for Security: 10th International Conference, GameSec 2019, Stockholm, Sweden, October 30–November 1, 2019, Proceedings 10*, Springer, 2019, pp. 479–491.

[75] R. E. Navas, F. Cuppens, N. B. Cuppens, L. Toutain, and G. Z. Papadopoulos, "Mtd, where art thou? a systematic review of moving target defense techniques for iot", *IEEE internet of things journal*, vol. 8, no. 10, pp. 7818–7832, 2020.

[76] K. Zeitz, M. Cantrell, R. Marchany, and J. Tront, "Designing a micro-moving target ipv6 defense for the internet of things", in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, 2017, pp. 179–184.

[77] A. Judmayer, J. Ullrich, G. Merzdovnik, A. G. Voyiatzis, and E. Weippl, "Lightweight address hopping for defending the ipv6 iot", in *Proceedings of the 12th international conference on availability, reliability and security*, 2017, pp. 1–10.

[78] C. Feng, Y. Zhang, and X. Wang, *Fedef: A federated defense framework using cooperative moving target defense*, IEEE, 2023. arXiv: 2306.11828 [cs.CR].

[79] E. T. M. Beltrán, Á. L. P. Gómez, C. Feng, *et al.*, "Fedstellar: A platform for decentralized federated learning", *Expert Systems with Applications*, vol. 242, p. 122 861, 2024.

[80] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database", in 2010.

[81] M. W. Gardner and S. Dorling, *Artificial neural networks: an introduction*. Springer Science & Business Media, 1998.

[82] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[83] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[84] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms", *arXiv preprint arXiv:1708.07747*, 2017.

[85] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images", 2009.

[86] A. G. Howard, M. Zhu, B. Chen, *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications", *arXiv preprint arXiv:1704.04861*, 2017.

Abbreviations

ML	Machine Learning
CNN	Convolutional Neural Network
MLP	Multilayer Perceptron
FL	Federated Learning
CFL	Centralized Federated Learning
DFL	Decentralized Federated Learning
IoT	Internet of Things
MTD	Moving Target Defense
PNR	Poisoned Node Ratio
PSR	Poisoned Sample Ratio
NR	Noise Ratio
IID	Independent and Identically Distributed
IID	Independent and Identically Distributed

List of Figures

List of Tables

Listings