University of Zurich[UZH]

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

A Secure Aggregation Protocol for Decentralized Federated Learning

Janosch Baltensperger
Zurich, Switzerland
Student ID: 15-915-085

Supervisor: Chao Feng, Dr. Alberto Huertas Celdran
Date of Submission: August 8, 2023

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/

Abstract

Poisoning attacks pose a substantial threat to the trustfulness of Federated Learning. For example, malicious participants can degrade the model performance of honest members or implement backdoors that can be exploited at inference time to take advantage of incorrect predictions. Researchers have been highly active to mitigate poisoning attacks. Existing approaches prominently aim for defenses against poisoning attacks in centralized settings. While decentralized Federated Learning has gained significant attention as a promising approach without a central entity, the security aspects related to poisoning attacks remain largely unaddressed.
This work introduces a defense approach called "Sentinel" for mitigating poisoning attacks in horizontal, decentralized Federated Learning. Sentinel leverages the advantage of local data availability and defines a three-step aggregation protocol composed of similarity filtering, bootstrap validation and normalization to protect against malicious model updates. The proposed defense mechanism is evaluated on various datasets under different types of poisoning attacks and threat levels. An extension of Sentinel, called SentinelGlobal, is presented, which incorporates a global trust protocol to reduce computational complexity and further improve the effectiveness against adversaries. Both Sentinel and SentinelGlobal demonstrate promising results against untargeted and targeted poisoning attacks. Hence, this work contributes to the advances in research against poisoning attacks in decentralized federated systems. Additionally, the results of this work highlight the need for more sophisticated defense strategies against backdoor attacks, independent of the Federated Learning architecture.

Verfälschungsangriffe („Poisoning Attacks“) stellen eine erhebliche Bedrohung für die Vertrauenswürdigkeit des föderalen Lernens („Federated Learning“) dar. Böswillige Teilnehmer können beispielsweise die Modellleistung ehrlicher Mitglieder verschlechtern oder Hintertüren implementieren, die zum Zeitpunkt der Inferenz ausgenutzt werden können, um von falschen Vorhersagen zu profitieren. Forscher haben sich intensiv mit der Abschwächung von Verfälschungsangriffen befasst. Bestehende Ansätze zielen vor allem auf die Abwehr von Verfälschungsangriffen in zentralisierten Umgebungen ab. Während dezentralisiertes föderiertes Lernen als vielversprechender Ansatz ohne zentrale Instanz grosse Aufmerksamkeit erlangt hat, bleiben die Sicherheitsaspekte im Zusammenhang mit Verfälschungsangriffen weitgehend unbehandelt.
In dieser Arbeit wird ein Verteidigungsansatz namens „Sentinel“ zur Abschwächung von Verfälschungsangriffen in horizontalem, dezentralisiertem föderiertem Lernen vorgestellt. Sentinel nutzt den Vorteil der lokalen Datenverfügbarkeit und definiert ein dreistufiges Aggregationsprotokoll, das aus Ähnlichkeitsfilterung, „Bootstrap“-Validierung und Normalisierung besteht, um vor böswilligen Modellaktualisierungen zu schützen. Der vorgeschlagene Verteidigungsmechanismus wird an diversen Datensätzen unter verschiedenen Arten von Verfälschungsangriffen und Bedrohungsstufen evaluiert. Es wird eine Erweiterung von Sentinel, genannt SentinelGlobal, vorgestellt, die ein globales Vertrauensprotokoll implementiert, um die Rechenkomplexität zu reduzieren und die Wirksamkeit gegen Angreifer weiter zu verbessern. Sowohl Sentinel als auch SentinelGlobal zeigen vielversprechende Ergebnisse gegen ungezielte und gezielte Verfälschungsangriffe. Daher trägt diese Arbeit zu den Fortschritten in der Forschung gegen Verfälschungsangriffe in dezentralen föderierten Systemen bei. Darüber hinaus unterstreichen die Ergebnisse dieser Arbeit den Bedarf an ausgefeilteren Verteidigungsstrategien gegen Backdoor-Angriffe, unabhängig von der föderalen Lernarchitektur.

Acknowledgments

First, I would like to thank Chao Feng, Ph.D. student at the Communication Systems Group at the University of Zurich, and Alberto Hueartas, postdoctoral researcher at the Communication Systems Group at the University of Zurich, for supervising my thesis. During regular meetings, they provided valuable inputs and critical feedback to keep me on the right track. Further, I would like to thank Prof. Dr. Stiller for allowing me to write my thesis at the Communication Systems Group.
Furthermore, I would like to express my gratitude to Enrique Tomás Martinez Beltrán, a Ph.D. student at the University of Murcia, for providing me with early access to Fedstellar and for his technical assistance.

Contents

Chapter 1

Introduction

Although the success of Machine Learning (ML) has risen exponentially in recent years, many domains cannot benefit from its advantages due to increased concerns about data privacy, compromised confidentiality, and limited access to compute resources [1]. Therefore, Federated Learning (FL) has emerged as a promising ML concept that enables different devices or organizations to collaboratively learn a model without submitting private data to a centralized server [2]. FL has the potential to solve privacy concerns, reduce latency, and increase scalability and reliability by distributing the learning tasks throughout many participants in a shared network [3]. Ultimately, the central server becomes solely responsible for model aggregation and broadcasting.

However, the traditional concept of FL is relying on a central entity, which implicates certain drawbacks such as communication bottlenecks and a single point of failure. For that reason, Decentralized Federated Learning (DFL) has been proposed as an alternative to avoid these vulnerabilities [4]. Despite the advantages and drawbacks of both FL architectures, research has shown that distributed ML in general is vulnerable to adversarial attacks, particularly those focused on manipulating training data [5]. FL is highly susceptible to these attacks, as data inspection techniques cannot be used to detect altered data. Hence, a dilemma exists between protecting privacy and establishing trust in FL. This vulnerability has led to an arms race between newly proposed defense mechanisms and corresponding attack strategies.

1.1 Motivation

One of the most significant threats to the integrity and effectiveness of FL are poisoning attacks [6]. The consequences of successful poisoning in FL can be severe: a compromised model may produce inaccurate predictions, misclassify inputs, or even introduce backdoors that adversaries can exploit later. Moreover, these attacks can intensively impair the collaborative learning process, eroding trust among participants and undermining the very essence of FL's privacy and security benefits. To defend against poisoning attacks in FL, researchers and practitioners have been actively exploring various robust and secure

algorithms and techniques. These defenses focus on detecting and mitigating the effects of poisoning attacks to maintain the integrity of FL and preserve the performance of the individual participants' model. However, most previous works focus on the security of Centralized Federated Learning (CFL), whereas defense techniques in DFL remain largely unexplored. In this context, this work aims to investigate the behavior of poisoning attacks in the DFL architecture and explore new defense strategies for increased security.

1.2 Description of Work

This work focuses on mitigating poisoning attacks in horizontal DFL. A new defense approach named Sentinel is presented, which takes advantage of the local data availability and defines a three-phase aggregation protocol to identify potentially malicious models. The first step of similarity filtering uses the layer-wise average cosine similarity to reject highly suspicious models. The remaining updates are then aggregated based on the local bootstrap validation loss, whereas the aggregation weight is calculated using an adaptive loss distance mapping. In a final step, the trusted models are normalized using the local model norm as a threshold, to reduce the impact of potential stealth attacks. In summary, Sentinel leverages a hybrid defense strategy composed of anomaly detection and robust aggregation to mitigate poisoning attacks.
The proposed aggregation mechanism is evaluated on the MNIST, FMNIST and CIFAR10 datasets under data and model poisoning attacks in targeted and untargeted forms. Different threat levels are investigated, which includes the number of attackers and the amount of modified data. Furthermore, an extension of Sentinel is introduced: SentinelGlobal implements a global trust protocol to further increase the security of Sentinel. The extended mechanism uses the binary classification result of Sentinel as a local trust score, which is shared with the federated network at each round. The global trust is then calculated based on the average trusted neighbor opinion, which integrates the trust perspective of assumed benign neighbors on a target node at hand. With the incorporation of global trust, the computational complexity of Sentinel could be reduced significantly.
Moreover, this work also contributes to the research advances in DFL with an adopted, synchronous version of Fedstellar, which paves the way for more comparable security benchmarks in DFL. The adapted platform can be retrieved on GitHub[1].

1.3 Thesis Outline

The structure of this work is outlined as follows. First, Chapter 2 establishes the theoretical baseline and describes the fundamental concepts used in this work. Subsequently, previously proposed defense mechanisms for both architectures, CFL and DFL, are analyzed for their limitations in Chapter 3. Based on these findings, an improved defense mechanism is proposed together with an extended trust algorithm in Chapter 4. Furthermore, a broad selection of poisoning attacks is defined and clarified, which is further

[1]https://github.com/janousy/fedstellar-sync

used to evaluate the newly proposed attack mitigation algorithm. In the next Chapter 5, the technical specifications of the defense strategy and the underlying framework to perform DFL experiments are outlined. An extensive evaluation of the proposed defense mechanisms against specified poisoning attacks is given in Chapter 6. Lastly, Chapter 7 summarizes the findings of this work and proposes future opportunities.

Chapter 2

Background

The following sections will summarize the theoretical foundation of FL and elucidate the taxonomy of common poisoning strategies and objectives. Generally, FL encompasses a broad set of applications, algorithms, and characteristics. Various concepts are transferable between the different variations of FL, especially when it comes to security measures. Hence, a general overview of FL and its vulnerabilities is given in this chapter, although the focus of this work lies on DFL.

2.1 Federated Learning

Traditional machine learning models typically require vast amounts of centralized data to be collected and transferred to a central server for processing [7]. In this centralized approach, the data is collected to train a global model that can then be distributed back to the individual devices or nodes for use [8]. While this centralized approach has shown remarkable success, its application in practice is often limited due to privacy regulations and legal restrictions [5]. FL addresses these challenges by bringing the model training process to the data sources themselves, rather than centralizing the data. In this federated setting, the training process takes place locally on each individual data center, computer, or edge device, ensuring that raw data never leaves the data owner's control [1]. FL has found applications in various domains, including healthcare, finance, internet of things (IoT), and natural language processing [7].

2.1.1 Fundamental Algorithm

Federated Average (FedAvg), proposed by McMahan *et al.* [2], has established itself as the de facto standard algorithm for FL. As illustrated in Figure 2.1, its execution is distributed over a central server and the participating clients or nodes. FL with FedAvg is composed of several rounds of alternating local model training and global aggregation. In a first step, the server initializes a global model w_0 with random parameters or pre-trained weights and distributes it to a selected subset of nodes. Each node then performs the local model

training by updating the local model's parameters w using techniques like Stochastic Gradient Descent (SGD) or Batch Gradient Descent. After local training, each client sends its model parameters (weights and biases) to a central server (aggregator) without sharing the raw data. At any round t, the central server aggregates the model parameters received from all clients by computing their weighted average. The aggregation weight for each client's model is usually determined by the amount of data (n_k) it contributed to the training process. The averaged model parameters become the updated global model w_{t+1}, which is in turn send back to each client again at the next round. The FL process is iterated for a predefined number of rounds or until a convergence criterion is fulfilled. With this procedure, the central server obtains a model that has been trained on data from various locations without having direct access.

However, FedAvg also poses some challenges, such as dealing with heterogeneous data distributions, addressing communication issues, and handling devices with limited computational capabilities [3]. Moreover, the security of this approach has been rigorously discussed since its proposal. Previous works have demonstrated that FedAvg is highly susceptible to poisoning attacks, as the protocol does not consider adversarial participation [5], [6], [9].

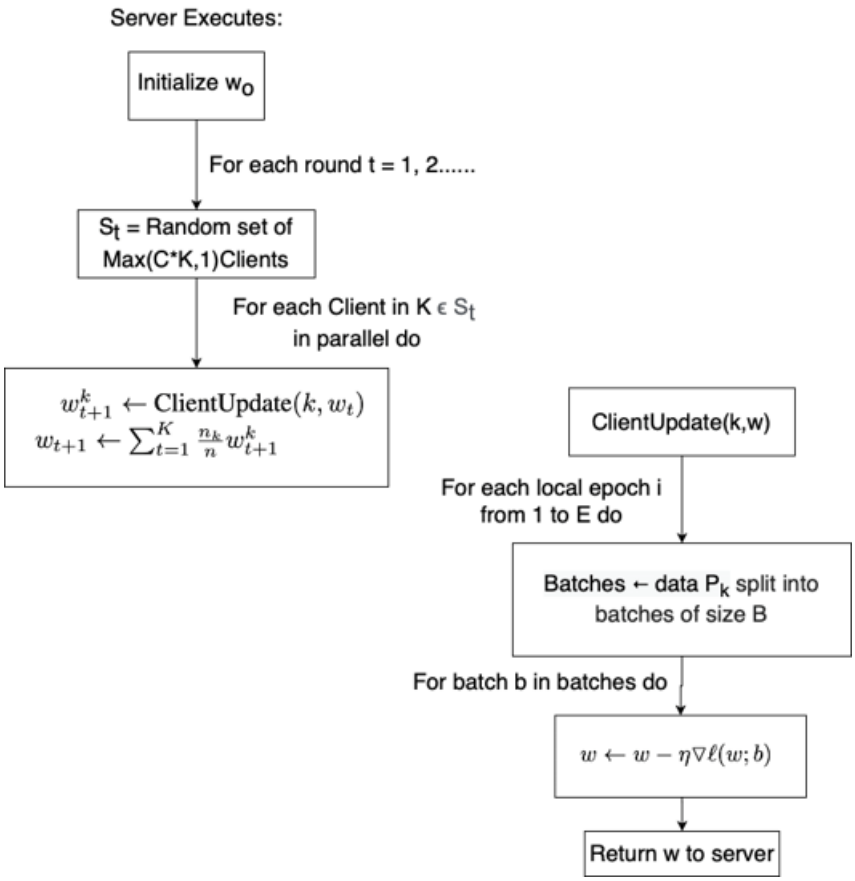


Figure 2.1: Federated Average (FedAvg) [10].

2.1.2 Architectural Variations

The architecture of CFL with a central entity has certain drawbacks [7]. First, all communication between clients and the central server is channeled through the network, whereby the server becomes a communication bottleneck with increased scale. Second, the central entity represents a single point of failure, since it is solely responsible for the orchestration of the FL process. Consequently, DFL has been proposed as an alternative, which attempts to avoid the architectural drawbacks of CFL [4]. In DFL, the central entity is eliminated, and its responsibilities are transferred to each local node. As a result, each participant alternately performs both tasks of local model training and aggregation of all model parameters received from its neighbors. The aggregation role can also be rotated between nodes at each round, resulting in a semi-decentralized architecture. Figure 2.2 illustrates these different FL architectures. In addition to serving as a trainer and/or aggregator, a node can also act as a proxy for model parameters to improve network connectivity [7].

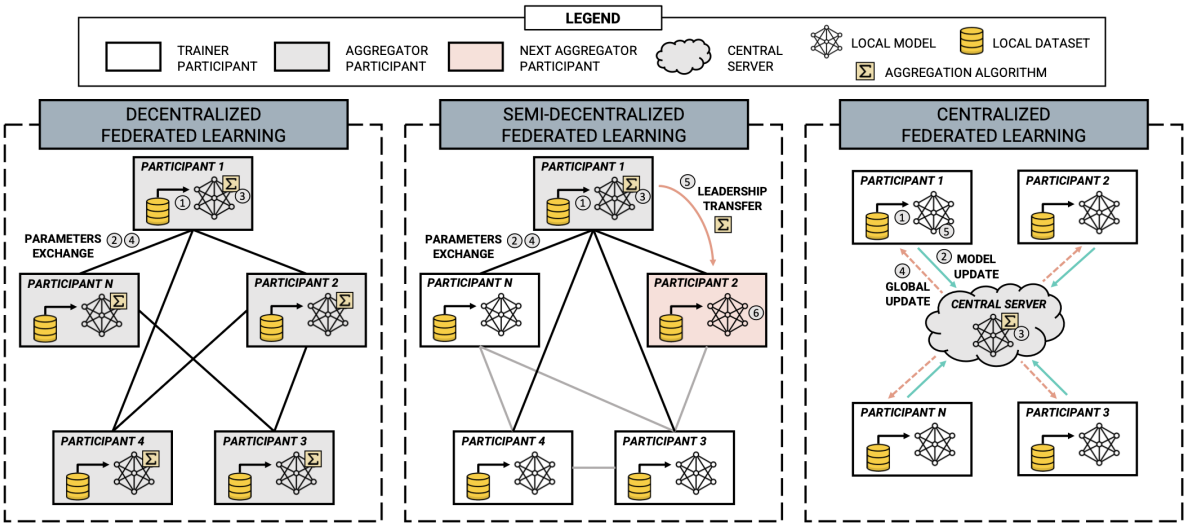


Figure 2.2: Variations in Federated Learning architectures [7].

Through this decentralization, the FL process can be fully autonomous and more resistant to participant failures. However, managing the dynamic node connectivity and orchestrating the learning process becomes increasingly complex. For example, the communication schema in FL can either be synchronous or asynchronous [11]. The former requires each node to wait for other nodes to complete their aggregation and then jointly proceed to the next round. In contrast, a node acts fully autonomously in an asynchronous schema. This can lead to the fact that the training process of some nodes converges faster, while stragglers fall behind. Moreover, each participant owns its individual local model instead of working towards a globally shared model, which may lead to a conflict of interest [12]. Furthermore, without the presence of a trusted central entity as in CFL, all participants are required to validate the honesty of their neighbors individually. Overall, these properties of DFL contribute to an increased threat by poisoning attacks.

2.1.3 Heterogeneity

The federated approach to ML presents novel challenges: besides communication efficiency, privacy, security and scalability, managing device and data heterogeneity is still an open research topic [10].
Device heterogeneity in a federated system originates from different hardware capabilities, including variations in processing power, memory, and storage capacities. For example, some devices may be high-end smartphones with powerful CPUs and large memory, while others may be low-end IoT devices with limited computational resources [13].
On the data level, individual nodes may have collected different types and scales of data with varying distributions. This is particularly significant when the training data is sourced from users in diverse geographical locations or with different usage patterns, which yields a shift in prior probability [14]. In FL, the assumption of independent and identically distributed (IID) samples across local nodes is often not realistic. Hence, tackling these distribution shifts is still actively investigated [15]. Furthermore, a non-IID scenario aggravates the challenge of differentiating between benign and malicious model updates [16]. This becomes especially important in DFL, where each local node trains towards a personal goal.

To tackle shifts in label or feature distribution, two popular variants of FL exist [7]: Horizontal Federated Learning (HFL) and Vertical Federated Learning (VFL). HFL is applied when participants posses a similar feature set but hold distinct data samples. VFL can be applied when multiple parties collaboratively train a federated model by combining their complementary data sets. Unlike HFL, the vertical variant involves parties with different sets of features but overlapping samples. The focus of this thesis lies on decentralized HFL.

2.2 Poisoning Attacks

The collaborative nature of FL makes the system a susceptible attack target. Model aggregators cannot validate other participants' contributions due to data privacy preservation. Attacks on FL can be classified into two main categories: poisoning attacks and privacy attacks [5]. Poisoning attacks are executed during the model training, whereas privacy attacks are conducted at inference time. The goal of poisoning attacks is to degrade the performance of the federated models or infiltrate their behavior. On the contrary, privacy attacks are executed by honest-but-curious actors with the aim of gathering sensitive information from other participants, *i.e.*, learning about their individually owned data.
The focus of this work lies on poisoning attacks. These can be viewed from two perspectives: the *attack strategy* defines how a poisoning attack is constructed, which is either through data or model modifications. In contrast, the *attack objective* specifies the goal of the adversary. Adversaries either have a specific target to be misclassified, or their attack is untargeted to degrade the overall model performance. The concepts of attack strategy and objective are discussed in detail in the following sections.

2.2.1 Attack Strategies

There exist two popular strategies to execute a poisoning attack: data and model poisoning [14], [17]. First, a data poisoning attack is performed at the data level. The adversary operates on its training data and thereby indirectly alters the ground-truth of the model training process. In contrast, model poisoning attacks presume a more relaxed scenario where clients can break the protocol and directly modify the locally trained model, *i.e.*, the submitted weights or gradients. These two concepts are illustrated in Figure 2.3. Note that the separation of data and model poisoning is continuous, meaning a poisoned model can be the result of a data poisoning attack [18].

Feasible attack strategies largely depend on an attacker's knowledge about the FL process, *e.g.*, whether they know about another client's data distribution. One can distinguish between full and partial attacker knowledge [19]: with full knowledge, the attacker is acquainted with the local datasets, protocols, and models of all participants. This is often considered practically infeasible, hence full attacker knowledge is not considered in this work. With partial knowledge, the attacker only has knowledge about compromised participants. In the following, the two strategies of data and model poisoning will be described in detail.

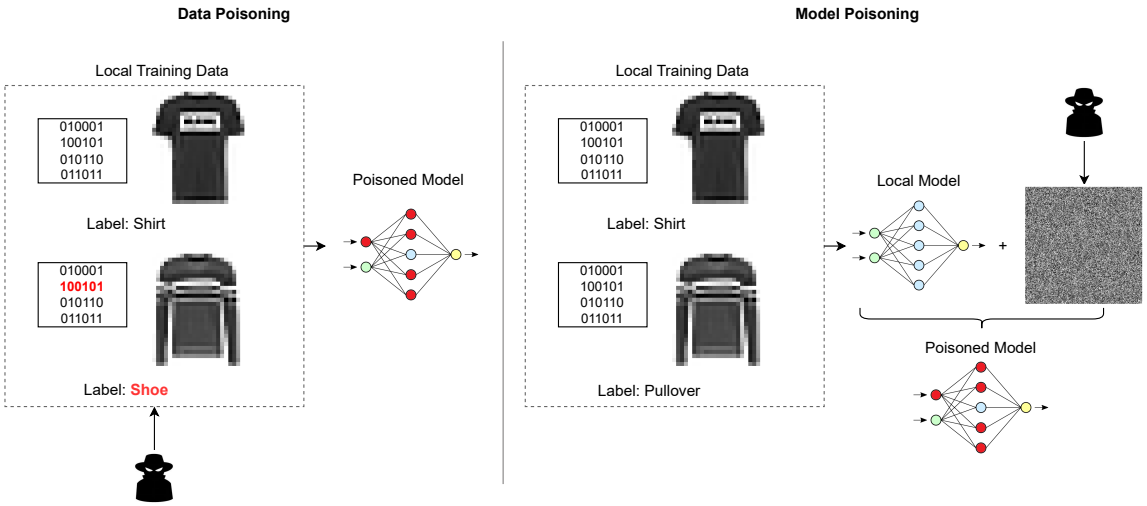


Figure 2.3: Illustration of data poisoning (left), where an adversary alters data labels and features, and model poisoning (right), with the example of added noise to the model weights.

Data Poisoning Attacks. The adversary manipulates its own local training data and thereby indirectly alters the source of the model training process. An adversary can achieve this through the manipulation of features or labels, or simply through data insertion. The main constraint of data poisoning attacks is their effectiveness. A large amount of poisoned samples and inherently numerous adversaries are required to make the attack successful [5]. Stand-alone data poisoning can be considered as offline attacks, as they are executed once before the training process is started [17]. The malicious actor then trains its model as usual, but on poisoned data. Data poisoning attacks can be combined

with model poisoning to maximize the adversarial impact [6]. The following are the two common data poisoning procedures used to attack FL models:

- Label Flipping [5], [20]: The adversary changes the labels of specific data records while leaving the features intact. Label flipping falls into the category of dirty-label attacks. The labels to be flipped are either chosen at random or selectively by a predefined target.
- Sample Poisoning [5], [20]: Instead of manipulating the labels of the data owned by an adversary, the sample's features are modified to deteriorate the class characteristics. This is also referred to as a clean-label attack since the labels remain unmodified. A common targeted strategy is the creation of artificial backdoors, discussed in Section 2.2.2.

Model Poisoning Attacks. Although data poisoning attacks eventually induce modifications to the model update, model poisoning attacks can be regarded as an individual attack class which does not interfere with the local training data [18], [21]. In contrast to data poisoning, model poisoning attacks presume a more relaxed scenario where clients can break the protocol and directly modify the locally trained model, *i.e.*, the submitted weights or gradients. Consequently, model poisoning attacks pose an enhanced threat, since they are more effective and usually harder to detect [6], [18]. However, model poisoning is less intuitive as this approach is not directly affiliated with the training data.

The categorization of model poisoning attacks is broad and there is yet a consensus to be reached in literature. In this work, it will be distinguished between two model poisoning attack strategies: *random weights generation* and *optimized methods*, whereas the two can also be combined [5].

- Random Weights Generation: The adversary directly manipulates the model weights by generating random values of the same dimensionality as benign updates [16]. A common strategy for this type of untargeted model poisoning is to add random noise to the local model update [6], [19].
- Optimized Methods: If the goal of the attacker is to remain undetected, a model poisoning attack becomes a trade-off between effectiveness and stealth [22]. Thus, adversaries aim to craft their update as similar as possible to past benign updates, while maximizing their attack impact. This strategy is usually employed to exploit the vulnerabilities of specific defense mechanisms [19], [23].

2.2.2 Attack Objectives

Poisoning attacks can be further categorized by attack objective, *i.e.*, as untargeted, targeted and backdoor attacks [6], [22]. These attack objectives can be realized through both, data and model poisoning. Note that in existing literature, there is not a strict but rather continuous distinction between these attack objectives [14]. Figure 2.4 gives an overview of different attack objectives using examples from object recognition.

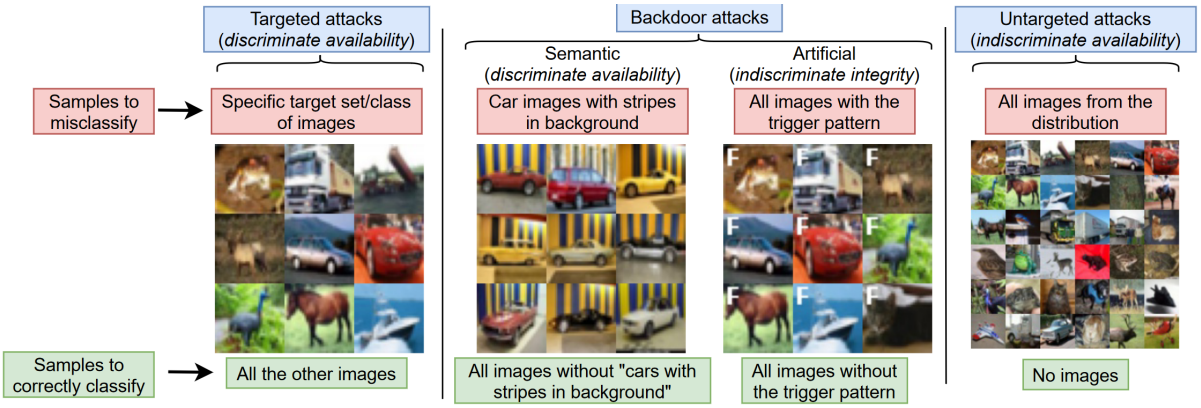


Figure 2.4: Different types of attack objectives in poisoning attacks: targeted, backdoor and untargeted [17].

Untargeted Attacks. The overall objective of untargeted attacks is to degrade the general model performance, *i.e.*, prevent the model from converging. Thus, this attack type can also be considered as an *indiscriminate availability attack* [17] or *Byzantine attack* [18]. As illustrated in Figure 2.4, none of the test data samples should remain correctly classified to consider an untargeted attack successful. Such attacks are often easier to detect than others, since the declined model performance is a good indicator for adversary presence [24], [25]. A common untargeted attack is sign-flipping [26], where an attacker simply inverts the algebraic sign of his updates. Further, the additive noise attack [26] using the aforementioned random weights generation strategy is another method of degrading model performance. Both of these approaches are model poisoning attacks. Data poisoning strategies such as random label flipping can also provoke an untargeted attack.

Targeted Attacks. In these attacks, an adversary deliberately manipulates the training data, with the goal of causing the model to produce incorrect or biased predictions on specific target inputs. The adversary infiltrates the model such that only a specific target set or class is misclassified, while the remaining set remains classified correctly [17]. Targeted attacks are usually more difficult to detect than untargeted, as they only affect the model accuracy of one specific class [27]. This target class is only known to the adversary. In Figure 2.4, a malicious actor could for example flip all labels belonging to the images of trucks to the label of a car. If the attack is successful, the poisoned model of a benign participant would incorrectly predict all trucks as cars. Inputs of other objects would remain classified correctly.

Backdoor Attacks. Another prevalent poisoning attack is the backdoor attack, which can also be regarded as a subclass of targeted attacks [5]. The goal is that only a specific target set with a certain property, *i.e.*, a trigger, is misclassified [17]. An adversary injects one or multiple triggers into the model during training, which can then be exploited during inference to the attacker's advantage [16]. Specifically, the model behaves normally in the absence of the trigger, but an attacker can provoke a desired prediction or classification by presenting the trigger at inference time. The trigger or target set property can be

of semantic or artificial nature [28]: (1) Semantic backdoors exhibit naturally occurring triggers, such as a striped background of certain colors (see Figure 2.4) or a specific word sequence. The trigger thus occurs naturally in the distribution of the training data, but the attacker flips the labels of these target instances. (2) On the other hand, artificial backdoors are crafted through manually injected triggers, *i.e.*, through sample poisoning. In Figure 2.4, this trigger is represented by an "F". To create an artificial trigger, the clean input data is modified before the FL process is started. However, these modifications do not necessarily have to be visible to the human eye. To prevent a backdoor from vanishing during the model averaging process, model poisoning techniques such as *constrain-and-scale* and *train-and-scale* have been developed to evade defense mechanisms and increase the model accuracy on the backdoor [16], [28].

Chapter 3

Related Work

In this chapter, the current state of research and the contribution of other authors is discussed. Due to the vulnerability of FL to poisoning attacks, researches have been highly active in investigating countermeasures. The most prominent methods are designed for the centralized setting, where the model aggregation by a single parameter server is less complex. In this work, the approaches to mitigate model poisoning attacks are categorized into: Byzantine-robust aggregation, anomaly detection and hybrid approaches [22], [28], [29]. In the following, each category and its encompassed techniques will be discussed.

3.1 Byzantine-Robust Aggregation

The main goal of these defense techniques is to provide a robust aggregation rule such that malicious updates cannot deteriorate the model performance. Thereby, convergence is ensured and the impact of the attacks is limited [23], [30]. This robustness is also referred to as Byzantine resilience [31]. These defense methods can mainly be divided into geometric measures, regularization, and decomposition. Geometric approaches can further be divided into vector-vise and coordinate-wise (or dimension-wise) filtering [17]. The latter is a more drastic approach, as it removes entire client vectors, instead of only the deviating vector elements. In general, Byzantine-robust aggregation methods defend well against untargeted poisoning but have limited effect against optimized, *i.e.*, stealthy, targeted attacks [19], [32].

Coordinate-wise Median. This defense method applies dimension-wise filtering [33]. The coordinate-wise median is a generalization of the median in higher dimensions, as represented in (3.1) for a single dimension j, where x_i represents a single client update. As a dimension-wise filtering approach is insensitive to skewed distributions, it effectively defends against a model replacement attack [16].

$$[\text{COMED}\,(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)]_j := \text{median}\left([\boldsymbol{x}_1]_j, \ldots, [\boldsymbol{x}_n]_j\right) \tag{3.1}$$

TrimmedMean. Similar to the coordinate-wise median, TrimmedMean [34] also applies dimension-wise filtering. However, the idea of TrimmedMean is to exclude the lowest and largest values in each dimension of the sorted updates using a trimming parameter, represented by β in (3.2). The foundational assumption of this approach is that malicious updates represent outliers when compared with honest updates. Both approaches, Median and TrimmedMean, are considered robust as long as the majority of participants is benign.

$$\left[\mathrm{TM}\left(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\right)\right]_j := \frac{1}{n-2\beta} \sum_{i=\beta+1}^{n-\beta} \left[\boldsymbol{x}_{j(i)}\right]_j \tag{3.2}$$

On the other hand, some approaches try to improve the robustness in CFL using a vector-wise approach, *e.g.*, RFA [35], Krum & Multi-Krum [36] or Bulyan [30].

RFA. The approach in RFA is a simple alteration of FedAvg, where the mean aggregation is replaced with the geometric median, as noted in (3.3). The geometric median is an effective technique against untargeted poisoning attacks, such as an additive noise attack. However, if an attacker's goal is to shift the geometric center with a sing-flipping attack, the defense can be broken [37].

$$\mathrm{GeoMed}\left(x_1, \ldots, x_k\right) := \arg\min_{y \in X} \sum_{i=1}^{k} \|y - x_i\|_+ \tag{3.3}$$

Krum. Next, a more sophisticated approach was proposed by Blanchard and his colleagues [36]. The core idea of Krum is to assign a score to each client update based on the similarity to other updates. Formally, a client's score is calculated by summing the Euclidean distance to the closest $n - f - 2$ other client updates, where n is the total number of clients, and f the amount of assumed adversaries. The global model is then updated with the vector from the client assigned to the lowest score. Thereby, the approach selects a single update that is most similar to a group of its closest neighbors, while excluding any potential malicious agents. The authors also proposed a variation called Multi-Krum [36]: instead of selecting a single client update as the next global model, m client updates are selected for averaging, where $1 \leq m \leq n$. The theoretical convergence guarantee of Krum & Multi-Krum is given for $n \geq 2f + 1$ and $n \geq 2f + 3$, respectively. However, the major limitation of Krum and other distance-based approaches is that they suffer from the curse of dimensionality, *i.e.*, the Euclidean distance becomes insignificant for large dimensions. An attacker can introduce larger perturbations for single model parameters without affecting the norm [25]. Additionally, an a priori assumption on the parameter f is required, *i.e.*, the number of malicious workers, which may be infeasible in practice [38].

Bulyan. Mhadi *et al.* [30] addressed the limitations of Krum with regard to the curse of dimensionality with Bulyan. First, the approach recursively selects $\theta \leq n - 2f$ model updates, *i.e.*, the closest neighbors, similarly to Krum. Second, a coordinate-wise Trimmed-Mean approach is applied to aggregate the selected set. Bulyan is reported to be robust for $n \geq 4f + 3$. However, experiments have demonstrated that in some scenarios, Bulyan performs worse than Multi-Krum due to potentially trimming benign updates [39].

All aforementioned statistical techniques based on vector-wise filtering build on the assumption that honest updates are close to each other, and malicious updates deviate in terms of distance. This assumption becomes problematic in the non-IID data setting, where honest updates may exhibit larger variance. Evaluations in the CFL architecture have shown that especially the Krum method suffered from diminished model accuracy when applied in a non-IID scenario [23], [27], [40]. Multi-Krum and Bulyan on the contrary have been reported to perform well under a non-IID data setting. Further comparisons have also demonstrated that more complex protocols are not necessarily performing better in certain attack configurations [5]. Fang *et al.* [41] evaluated different aggregation strategies, *i.e.*, TrimmedMean, Coordinate-wise Median, Krum and Bulyan, within their BRIDGE framework in the decentralized schema with Byzantine failures. Their evaluations support the reported performance drops as the data distribution is shifted towards extreme non-IID settings.

Zeno++. Xie *et al.* [11] proposed an asynchronous protocol for robust CFL that also applies regularization. The authors suggested an approximated gradient descent score based on the estimated loss and the update magnitude to decrease the impact of potentially malicious gradient updates. This score thus represents a client's trustworthiness. Consequently, the approximation requires that the server owns a validation dataset, which can be viewed as a major limitation in the centralized setting. Robustness based on performance instead of parameters also introduces a computational overhead [31]. Additionally, the author did not evaluate the protocol in non-IID scenarios. Nevertheless, Zeno++ has the advantage of exhibiting robustness against a large number of attackers, as the protocol only requires at least one benign update for convergence.

Adaptive Federated Averaging. Muñoz-González *et al.* [42] proposed an approach that does not just focus on filtering potentially malicious updates, but blocks adversaries entirely from participation. Adaptive Federated Averaging (AFA) uses a Hidden Markov Model based on gradient cosine similarity to predict a client's behavior, thereby introducing a learning component to the defense system. This predictive score is then used as a weight factor in the aggregation, representing a regularization term. The approach demonstrated increased performance in comparison to other techniques such as Multi-Krum or Coordinate-wise Median. Using cosine similarity has the advantage that it is not affected by intentionally scaled updates, as it is the case for Euclidean distance [43]. However, the evaluation did not regard non-IID scenarios or more advanced targeted poisoning attacks. For example, a malicious actor could pretend to be benign and only attack within the last few iterations to decoy the behavioral scoring [31].

RSA. Li *et al.* [26] incorporated the ℓ_1-norm regularization into SGD to increase the robustness against Byzantine attacks. The mechanism effectively defended against scaled sign-flipping attacks. Furthermore, the authors demonstrated increased performance in comparison to other Byzantine defense methods such as Krum or geometric median based approaches, especially in the non-IID settings. However, other evaluations debated these results [17].

DnC. Shejwalkar *et al.* [44] suggested an alternative defense mechanism for model poisoning attacks in CFL, called divide-and-conquer (DnC). The approach tackles the curse of dimensionality problem through dimensionality reduction, and then uses spectral analysis, *i.e.*, singular value decomposition, to detect and filter poisoned updates. Evaluations demonstrate that these techniques perform well in the IID setting. However, experiments on the non-IID FEMNIST dataset indicate that DnC cannot effectively prevent attacks when the adversary has knowledge about honest updates from benign clients. Despite this limitation, DnC has an advantage over other vector-wise filtering methods in terms of computational complexity, as the algorithm does not require a pair-wise distance matrix of each update vector.

Robust Learning Rate. As a defense against backdoor attacks, researchers also investigated a learning rate decomposition. In the context of CFL, the server can modify the learning rate per gradient dimension and thereby control the client's contributions [5]. Ozdayi *et al.* [24] suggested a lightweight strategy called robust learning rate (RLR). It builds on previous learning rate modification approaches such as SignSGD [45], which reportedly performed badly in the non-IID setting [17]. The key concept of RLR is that clients vote for a global model update direction through the algebraic sign of their update vector. For every dimension of the update vector, the total number of votes must reach a certain learning threshold, otherwise, the particular learning rate is negated. Thereby, the aggregation algorithm moves towards the gradient for dimensions where there is insufficient agreement, and thus moving away from potentially malicious updates. Evaluations have demonstrated, that RLR serves as an effective defense against backdoor attacks, regardless of the data distribution [5]. The drawback however is the choice of the learning threshold, which may not be exceeded by the number of malicious clients.

3.2 Anomaly Detection

Defense mechanisms through anomaly detection, also referred to as Byzantine detection, aim at identifying and removing potentially malicious updates [29]. In contrast to Byzantine robustness, these anomaly detection schemes do not implement the defense strategy into the aggregation rules. Such detection mechanisms can either be classified as validation-based or gradient-based. Validation-based approaches investigate the performance behavior of individual updates, whereas gradient-based methods analyze the inherent numeric properties.

ERR & LFR. Building on robustness guarantees in distributed machine learning, Fang *et al.* [19] proposed two adapted defense strategies for CFL: Error Rate based Rejection (ERR) and Loss Function based Rejection (LFR). The key concept of these strategies is to evaluate the performance of the collected client models on a server-side validation dataset. Before averaging the received models, a predefined number of updates that have the largest impact on either the loss or the validation error are rejected. Subsequently, the remaining models are aggregated using a Byzantine robust mechanism, *e.g.*, Median, Krum or TrimmedMean. The authors evaluated ERR & LFR individually and also both applied simultaneously for untargeted poisoning attacks. The defense mechanisms demonstrated limited effectiveness when an attacker has full-knowledge about the deployed defense. As a drawback, the server is required to collect a clean data set, which may violate the privacy-preserving concept of FL.

PDGAN. To overcome the limitation of server-side data collection, Zhao *et al.* [46] proposed PDGAN. Instead of requiring a clean validation dataset, the system trains a generative adversarial network (GAN) concurrently to the original federated learning task. Once trained, PDGAN reconstructs a client's data from their model updates received on the server side. The generated data can then in turn be used to audit individual updates and label malicious clients as attackers. Focusing on a heterogeneous data distribution, the authors reported an effective defense against label flipping attacks. However, the mechanisms can only defend from attackers after a certain number of iterations, when the GAN is assumed to be trained well-enough. Additionally, the server must be sufficiently performant to train a neural network in time and simultaneously to the federated aggregation.

FoolsGold. Fung *et al.* [40] proposed an anomaly detection mechanism based on gradient similarity called FoolsGold. It is designed to defend against Sybil attacks in CFL under the assumption that the clients controlled by an adversary exhibit a large similarity in comparison to benign updates. The authors introduce a client contribution similarity based on cosine similarity and add a historical component to the algorithm that analyses the variance of client updates over time. Thereby, FoolsGold identifies a Sybil group when a set of updates becomes too similar. Overall, the approach works well in the non-IID setting with a high number of adversaries. Evidently, the effectiveness decreases as the data distribution shifts towards IID and benign gradient updates exhibit higher similarity [23]. FoolGold is vulnerable to constrain-and-scale attacks, as there is no norm-thresholding defense applied [28]. Additionally, FoolsGold fails to defend against a single adversary due to the nature of the adversary classification. As a mitigation, the authors suggest to use FoolsGold in combination with other defense strategies. Due to the strong data distribution assumptions of FoolsGold, the method also fails to defend against simultaneously injected backdoors and stealthier constrained attacks [43].

FLDetector. Another approach to detect malicious participants is to investigate the gradient consistency of client updates. FLDetector [47] predicts a client's update based on its past contributions. Specifically, the server applies a quasi-newton approach to estimate

the Hessian of an update and compare it to the integrated Hessian. The defense appears to be effective against various adaptive attacks, such as scaling attacks, distributed backdoors and untargeted model poisoning. However, predicting consistency is computationally expensive.

Spectral Anomaly Detection. Li *et al.* [37] suggested a defense mechanisms based on training a spectral anomaly detection model to identify malicious updates in a low-dimensional latent space. Essentially, an encoder-decoder architecture is trained on unbiased model updates. The encoder creates low-dimensional embeddings that represent data variability in absence of noise and redundant features. The decoder then aims at reconstructing a model update from these embeddings. The reconstruction error arising from this process allows differentiating between benign and malign updates. The latter exhibit a larger reconstruction error under the assumption that an attacker modifies their update. The authors propose to dynamically set the error threshold based on the mean error. Consequently, updates above the threshold are excluded from the aggregation, and the updates considered benign are weighted based on their reconstruction error. Benchmarks in a non-IID environment demonstrated an effective defense against untargeted and targeted attacks, *i.e.*, sign-flipping, additive noise and artificial backdoors. However, appropriately selecting the benign model updates to train the encode-decoder model is critical.

Sniper. Based on gradient similarity, Sniper [48] maps client updates into a graph-cluster. For each pair of updates, the Euclidean distance is calculated and an edge between the two clients is created if the distance is below an adaptive threshold. If there exists a maximum clique composed of more than half of all clients, the updates of this clique are aggregated. Otherwise, the threshold limiting the edge creation is increased. The method effectively defends against untargeted and targeted attacks, as long as the data is IID. In a more heterogeneous setting, the performance declines.

3.3 Hybrid Defense Techniques

In this work, hybrid defenses are defined as approaches that employ a combination of both, robust aggregation and anomaly detection mechanisms.

FLTrust. Cao *et al.* [32] proposed a trusted aggregation mechanism for CFL based on comparing local model updates to a trusted bootstrapping model. Therefore, the server acquires a clean root dataset to train a reference model on its own. At each iteration, the received model updates are compared to the reference model using the ReLU-clipped cosine similarity. The result is then used as a trust score, whereby updates that deviate too much from the reference model receive a score of zero. Additionally, each client update is normalized using the norm of the reference model as a scaling factor. All normalized updates are then aggregated using their trust score as a weight. The authors

evaluated FLTrust in homogeneous and heterogeneous distributions and demonstrated that the method effectively defends against untargeted local model poisoning, backdoors through label-flipping and scaled targeted attacks. Other experiments [49] debated this and indicate high attacker success rates with increasing non-IID degree. Furthermore, it may not be practicable in a CFL architecture to obtain a root dataset.

Trusted DFL. Gholami *et al.* [50] implemented the concept of trusted aggregation in the decentralized architecture. Each node is judged based on a behavioral score that reflects a participant's performance contribution and update consistency. Computationally, this is represented by cluster-based and distance-based metrics. The local trust score of each neighboring node is then broadcasted, such that each node can compute a global trust score based on their neighbors opinions. Thereby, the trust perspective of each node on their neighbors propagates through the network. At each iteration, local trust and global trust scores are recomputed, which allows a node to recover from misjudgment. Finally, the global trust score of a neighboring node serves as a weighting factor during the aggregation phase. This trust framework has only been successfully evaluated on untargeted model attacks (random weights) with a low number of attackers ($\leq 20\%$) in a non-IID scenario. The defense effectiveness on targeted attacks, *e.g.*, artificial backdoors, is yet to be established. However, the authors suggest using their framework as a building block for secure DFL.

FedInv. Zhao *et al.* [49] tackled the limitations of anomaly detection mechanisms such as FLTrust and PDGAN requiring training data. The defense mechanism is inspired by membership inference attacks. The authors proposed a privacy-preserving model inversion strategy, such that dummy data can be synthesized from each client's gradient updates. In a second step, the local model updates are assigned a score: for each pair of synthesized dummy dataset, the distribution divergence is calculated using the Wasserstein distance. To reflect non-IID scenarios, the scores are additionally clustered into two groups. The group containing the majority of participants is then aggregated. The authors evaluated the approach on various datasets and varying data distributions. In comparison to Multi-Krum, FLTrust and LFR and tested against untargeted (Gaussian noise, Krum attack) and targeted attacks (Badnet, label flipping), FedInv demonstrated the best performance overall. The authors claimed that this was the first defense approach to mitigate stealthy poisoning attacks. However, the computational complexity of FedInv was discussed. Moreover, although the authors claimed that the model inversion is privacy-preserving, reconstructing client data may contradict the concept of federated learning.

Norm Clipping. A thoroughly investigated approach against targeted attacks is vector re-scaling, also referred to as clipping, norm bounding or thresholding [39]. Norm clipping is designed to defend against semantic backdoor attacks combined with boosting in CFL, where an attacker performs label flipping and submits update vectors with scaled norms for greater attack impact [24]. Thus, the idea is to limit the scale of individual model weights using a threshold parameter M, as defined in (3.4). The model weights are represented by

x_t^k, where t indicates the aggregation round and k the client index in the set of participants S. Applying norm thresholds to model updates provably limited the success of naive model poisoning attacks such as scaling [28]. However, the approach was not effective against untargeted poisoning such as sign-flipping [37]. The defense mechanism only considers the magnitude, but not the direction of an update. Additionally, finding a suitable clipping threshold imposes another difficulty.

$$\Delta x_{t+1} = \sum_{k \in S_t} \frac{\Delta x_{t+1}^k}{\max\left(1, \left\|\Delta x_{t+1}^k\right\|_2 / M\right)} \tag{3.4}$$

Subsequently, Karimireddy *et al.* [51] proposed a centered clipping strategy, given in (3.5): The local model updates x_i are compared to the global model v_l of the previous aggregation at round $l+1$, then the norm is limited by a threshold parameter τ_l. Note that the threshold is set iteratively. The authors additionally introduced worker momentum to tackle time coupled attacks that introduce an accumulated error over time. By using momentum, the clients' gradients are averaged to reduce the variance. A worker momentum is composed of the momentum of the previous round $m_{t-1,i}$ and the current gradient $g_i(x_{t-1})$, weighted by a momentum β, as display in (3.6). The server then aggregates all worker momenta. The authors empirically evaluated their defense strategy in an IID scenario and determined the momentum β and the clipping threshold τ empirically based on the learning rate. Experiments demonstrated an effective defense against untargeted and backdoor attacks at low computational costs. Furthermore, He *et al.* [52] transferred the idea of centered clipping and local worker momentum to the DFL architecture. Instead of using the global model as a reference, the local model update is used, which yields *i.e.*, self-centered clipping. Evaluations show that the defense strategy also performs well in non-IID scenarios. However, the authors argue that choosing a suitable clipping threshold is still an open problem.

$$v_{l+1} = v_l + \frac{1}{n} \sum_{i=1}^{n} (x_i - v_i) \min\left(1, \frac{\tau_l}{\|x_i - v_l\|}\right) \tag{3.5}$$

$$\boldsymbol{m}_{t,i} = (1 - \beta_t)\, \boldsymbol{g}_i\left(\boldsymbol{x}_{t-1}\right) + \beta_t \boldsymbol{m}_{t-1,i} \tag{3.6}$$

DeepSight. Rieger *et al.* [23] proposed a rather unique defense strategy called DeepSight for CFL. The authors argue that using a combined strategy of filtering and clipping drastically reduces an attacker's possibilities. It is argued that existing hybrid approaches often misclassify benign updates. Thus, DeepSight analyses the internal neural network structure of each client update. By investigating the last layer, the author's defense creates a first fingerprint based on label distribution and a second fingerprint based on the change in predicted probability. To generate the fingerprints, the approach does not require any server-side data. Instead, random input vectors are used as model inputs. Additionally, the pair-wise cosine similarity is calculated. These three characteristics then serve as features of the defense layer. On one hand, HDBSCAN is used to cluster updates. On the other hand, the models are classified based on backdoor probabilities. Finally, the filtered

models are clipped before aggregation. DeepSight serves as an effective defense in both IID and non-IID scenarios against sophisticated backdoor attacks. The disadvantage of DeepSight is its computational complexity. Especially, the network fingerprinting creates a significant overhead.

FLAME. With the goal to mitigate backdoor attacks, Nguyen *et al.* [43] propose a hybrid approach based on dynamic clustering, adaptive clipping and noising. In a first step, the client updates are clustered based on pair-wise cosine similarity to capture angular deviation. The authors argue that existing clustering-based approaches often group the models into malicious and benign. This leads to the issue that when no adversaries are present, benign updates become removed. Therefore, a custom HDBSCAN clustering algorithm is applied under the assumption, that at least half of the participating nodes are benign. Updates considered as outliers are then filtered. In a second step, the updates are clipped to reduce the impact of potentially undetected malicious updates. The authors argue that generally, the average norm of updates decreases as the training process proceeds. Therefore, a dynamic clipping threshold is proposed, whereby at each iteration, the median norm of all updates (incl. outliers) is set as the norm threshold. As a final step, the aggregated model is smoothed using adaptive noising. Broadly, the amount of noise is determined based on the distances of the remaining model updates. The authors evaluated FLAME for various datasets under backdoor and untargeted attacks with success. However, with the clustering configuration in use, the defense fails with the number of attackers rising above 50%. Additionally, with the data distribution shifting towards extreme non-IID cases, the model suffers from slight performance drops. As HDBSCAN is computationally expensive, FLAME cannot be regarded as a light-weight approach.

3.4 Post-Aggregation Techniques

In the context of this work, post-aggregation techniques are defined as approaches that do not interfere with the aggregation progress. Instead, model updates are modified after the aggregation.

Neuron Pruning. As a defense against backdoor attacks in the CFL setting, Wu *et al.* [53] presented a technique based on neuron pruning. The underlying concept of pruning is that backdoors activate neurons of a neural network that would otherwise remain dormant, *i.e.*, not activated. To identify the maliciously activated neurons, each client submits a voting sequence to the server, based on their averaged activation values of the last network layer from the locally trained model. The server then aggregates these votes and decides which neurons to prune. Pruning is only effective for targeted attacks, as untargeted attacks affect the entire neural network [17].

Weak Differential Privacy. Sun *et al.* [39] additionally proposed the incorporation of weak differential privacy in the form of Gaussian noise for extended robustness. Smoothing

model updates is a common technique to achieve differential privacy. The authors argue that low amounts can also protect against backdoor attacks. However, there is a trade-off between defense effectiveness and negative impact on model accuracy. When applied as a stand-alone defense mechanism, the main task accuracy drops significantly. It is also not trivial to determine the amount of noise to be added. Nevertheless, the approach is used as an effective defense component in various works [43], [54].

3.5 Research Motivation

As of the current state of research, a noticeable knowledge gap exists in the domain of DFL concerning defenses against poisoning attacks. Poisoning attacks pose a substantial threat to the trustfulness of collaborative learning. Previous attempts prominently explored defenses against poisoning attacks in centralized settings, but only few works investigated defense mechanisms in fully decentralized systems. While DFL has gained significant attention as a promising approach without a central entity, the security aspects related to poisoning attacks remain largely unaddressed.

A summary of previous work with a focus on either DFL or CFL is available in Table 3.1. This overview highlights the need for innovative research and robust techniques to safeguard DFL systems against poisoning attacks effectively. For comparison, the defense mechanisms proposed in this work are also listed. Defense approaches that incorporate the use of Blockchain solutions are not considered, since the goal is to design a lightweight DFL protocol without the dependency on an additional decentralized system. Moreover, most works assume that the majority of the FL participants act honestly. Consequently, this work will investigate the mitigation of poisoning attacks in DFL without any assumptions about the adversarial environment. Addressing this research gap will pave the way for secure and trustworthy FL in decentralized environments.

Table 3.1: Classification of defense techniques against model poisoning attacks. For each approach, the intended effectiveness against untargeted (U) and targeted attack (T) is reported, unknown effectiveness is marked with "?". The schema describes the FL architecture for which the defense method has been designed for: CFL or DFL. Adopted from [22], [28], [29].

Category	Type	Method	Technique	Schema	Objective	
					U	T
Robust Aggregation	Geometry	COMED [33]	Coordinate-wise median	CFL	✓	x
		RFA [35]	Geometric median	CFL	✓	x
		TrimmedMean [34]	Filtered mean	CFL	✓	x
		Krum [36]	Euclidean distance	CFL	✓	x
		Multi-Krum [36]	Euclidean distance	CFL	✓	x
		Bulyan [30]	Krum and TrimmedMean	CFL	✓	x
	Regularization	Zeno++ [11]	Approximated gradient descent score	CFL	✓	x
		AFA [42]	Gradient similarity, Hidden Markov model	CFL	✓	x
		RSA [26]	Norm regularization	CFL	✓	x
	Decomposition	DnC [44]	Dimensionality Reduction (PCA) and SVD	CFL	✓	x
		RLR/SignSGD [24], [45]	Learning rate decomposition	CFL	?	✓
Anomaly Detection	Validation	ERR, LFR [19]	Global validation	CFL	✓	✓
		PDGAN [46]	Model accuracy auditing	CFL	✓	✓
	Gradient-based	FoolsGold	Gradient similarity (cosine)	CFL	✓	✓
		FLDetector [47]	Hessian-based gradient consistency	CFL	✓	✓
		Li et al. [37]	Spectral anomaly detection	CFL	✓	✓
		Sniper [46]	Graph clustering	CFL	?	✓
Hybrid Mechanism		FLTrust [32]	ReLU-clipped cosine similarity, norm thresholding	CFL	✓	✓
		Gholami et al. [50]	Trusted aggregation	DFL	✓	?
		FedInv [49]	Gradient-based clustering distribution divergences	CFL	✓	✓
		Centered Clipping [52]	Clipping and worker momentum	DFL	✓	✓
		DeepSight [23]	Classification and clustering with update fingerprinting	CFL	✓	✓
		FLAME [43]	Clustering (cosine similarity), adaptive clipping, noising	CFL	✓	✓
Post-Aggregation		Wu et al. [53]	Neuron Pruning	CFL	x	✓
		Sun et al. [39]	Weak Differential Privacy	CFL	x	✓
This work		Sentinel	Model similarity, bootstrap validation and normalization	DFL	✓	✓
		SentinelGlobal	Trusted neighbor opinion	DFL	✓	✓

Chapter 4

Defense Design

With the theoretical baseline and the identified research gap established, this chapter introduces the fundamental design choices of poisoning attacks and defense approaches used in this work. First, a set of attacks is selected that will serve as a measure of defense effectiveness in subsequent chapters. Second, the underlying concepts of the defense strategies proposed in this work will be thoroughly discussed.

4.1 Attack Specification & Evaluation Metrics

In the context of this work, four types of poisoning attacks are chosen to analyze the behavior of existing and newly proposed defense techniques: targeted and untargeted label flipping, targeted sample poisoning and untargeted model poisoning. Further on, the term backdoor attack and targeted sample poisoning is used interchangeably. This set of attacks is selected as it represents the baseline performance measure of defense mechanisms against poisoning attacks. Furthermore, these attacks are not designed to make any assumptions about the attacker's knowledge of the deployed aggregation algorithm. Additionally, the following attack parameters are introduced:

- Poisoned Node Ratio (PNR) describes the percentage of participants in the network which are malicious.
- Poisoned Sample Ratio (PSR) represents the percentage of data samples (features) or labels altered by the attacker.
- Noise Ratio (NR) corresponds to the amount of noise an attacker introduces to the local data or model.

Table 4.1 gives an overview of all attacks and their parameters. Not all parameters are applicable to an attack. The details of the individual implementation of each attack will be clarified in the course of this section.

Attack Type	Parameter			Metric
	PNR [%]	PSR [%]	NR [%]	
Model Poisoning	[0,100]	-	[0,100]	F1-Score
Untargeted Label Flipping	[0,100]	[0,100]	-	F1-Score
Targeted Label Flipping	[0,100]	[0,100]	-	ASR_{lf}
Backdoor	[0,100]	[0,100]	-	BA

Table 4.1: Configuration overview for each of the selected attacks. Cells marked with "-" specify that the metric is not applicable to the corresponding attack.

Untargeted Model Poisoning. For this attack, a random weights generation strategy is applied, as described in Section 2.2.1. Specifically, the attack uses a salt noise, which changes a selected weight to the value 1. The number of weights to be altered is defined by the NR $\in [0, 100]$. Thus, adversaries train their local model on the original data, and apply model poisoning before sending the model to their neighbors. Model poisoning is applied at each round of the FL process. As the goal of untargeted attacks is to degrade the model performance of benign participants, the Attacker Success Rate (ASR) is measured by the average F1-score of all benign participants. The F1-Score is defined in (4.1). The lower the average benign F1-score, the more successful the attack.

$$\text{F1} = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4.1}$$

Untargeted Label Flipping. For this type of attack, the adversary changes the labels of any local data items from the original label l_i to a randomly chosen label l_{mod}, with $l_i, l_{mod} \in L$, where L is the set of all labels known to the attacker. The number of labels to be flipped is defined by the PSR, where PSR $\in [0, 100]$. Therefore, an attacker can flip 100% of the local labels at maximum. The sample features remain unmodified. The aim of this attack is to degrade the model performance of benign actors. Thus, the ASR of untargeted label flipping can be measured by the average benign F1-score, similar to untargeted model poisoning.

Targeted Label Flipping. With the targeted objective of label flipping, the attacker aims at provoking a misclassification of a source label to a desired target label. Therefore, the labels of the local data items are not chosen randomly. Instead, the adversary changes data items having a source label l_{src} to a predefined target label l_t, with $l_{src}, l_t \in L$. Local items with labels other than the source label remain unmodified. The attacker can choose to flip the source labels at a PSR $\in [0, 100]$. The ASR of targeted label flipping is measured by the number of samples having a true label $y = l_{src}$ being incorrectly predicted as the target label, *i.e.*, $\hat{y} = l_t$. With the confusion matrix C computed on the local test data, the ASR of targeted label flipping is defined as in (4.2), where c_{ij} is the number of

samples having true label y_i and predicted label $\hat{y}_j$. L represents the set of labels in the corresponding dataset. The ASR is averaged for all benign participants.

$$ASR_{lf} = \frac{c_{src,t}}{\sum_{j=0}^{|L|} c_{src,j}} \tag{4.2}$$

Targeted Sample Poisoning. The goal of the targeted sample poisoning is to introduce a backdoor in the model of the benign participants. In this work, only artificial backdoors are considered. Specifically, the adversary marks local data samples corresponding to a specific target label l_t with a trigger in the form of a 5×5 pixel X on the top-left of an image, as illustrated in Fig. 4.1. Similarly, the attacker chooses any PSR $\in [0, 100]$. To measure the backdoor accuracy (BA), a backdoor dataset B is constructed, where all samples of any label are marked with the defined trigger. B has the same size as the local test dataset. The BA is defined as the ratio of samples incorrectly predicted as the target label: with the confusion matrix C computed on the backdoor data using an individual local benign model, the BA is defined as in (4.3), where c_{ij} is the number of samples having true label y_i and predicted label $\hat{y}_j$. L represents the set of labels in the corresponding dataset. The BA is then averaged for all benign participants.

$$BA = \frac{\sum_{j=0}^{|L|} m_{j,t} - m_{t,t}}{|D| - m_{t,t}} \tag{4.3}$$

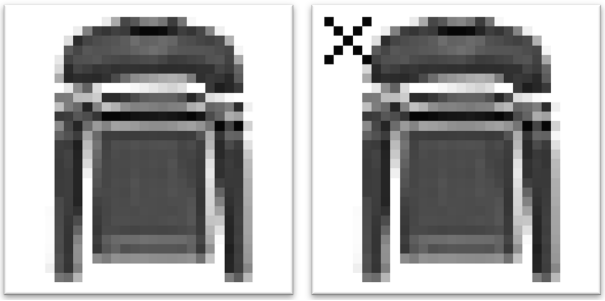


Figure 4.1: An example of an original low-resolution image of a sweatshirt (left), and the same image with the artificial trigger added on the top-left (right).

4.2 Defense Design

In CFL, the nodes are organized in a star network with a server as the central entity and there are no connections between the individual nodes. It is therefore feasible to assume that a defined subset of models is received on the server at each round. This property allows for the application of grouping algorithms into benign and malicious nodes. However, this concept does not hold for DFL. Due to the dynamic connectivity and diverse network topology, there is no fixed number of neighbors for an individual node. As a consequence, commonly used anomaly detection mechanisms, such as clustering, are

not applicable to detect adversaries. Instead, a mechanism is defined that does not rely on the number of neighbors. On the other hand, defense mechanisms in DFL have the advantage of local data availability that can be used to evaluate models received from neighbors. In the following sections, a defense mechanism called Sentinel will be proposed that adopts the aforementioned properties of DFL. Furthermore, an extension of Sentinel, SentinelGlobal, will be proposed that aims to reduce the computational complexity and increase the stability of the base aggregation algorithm.

4.2.1 Sentinel

Sentinel is a defense mechanism that aims to adapt to the dynamic connectivity DFL and does not depend on the number of neighbors in each round. The purpose of Sentinel is to defend against poisoning attacks. It incorporates the local data availability and relies on the two metrics cosine similarity and bootstrap loss to evaluate the trust performance of a neighbor's model. Sentinel, defined in Algorithm 1, is a three-phase protocol consisting of (1) similarity filtering, (2) bootstrap validation and (3) layer normalization. These steps will be elucidated in the following paragraphs. An illustration of the protocol is given in Figure 4.2. The formal overview of Sentinel is given in Algorithm 1. If not otherwise stated, the local model M is defined as the model trained on the individual node available in any round when Sentinel or SentinelGlobal is applied. A model received from an adjacent node n_i is referred to as a neighbor model P_i.

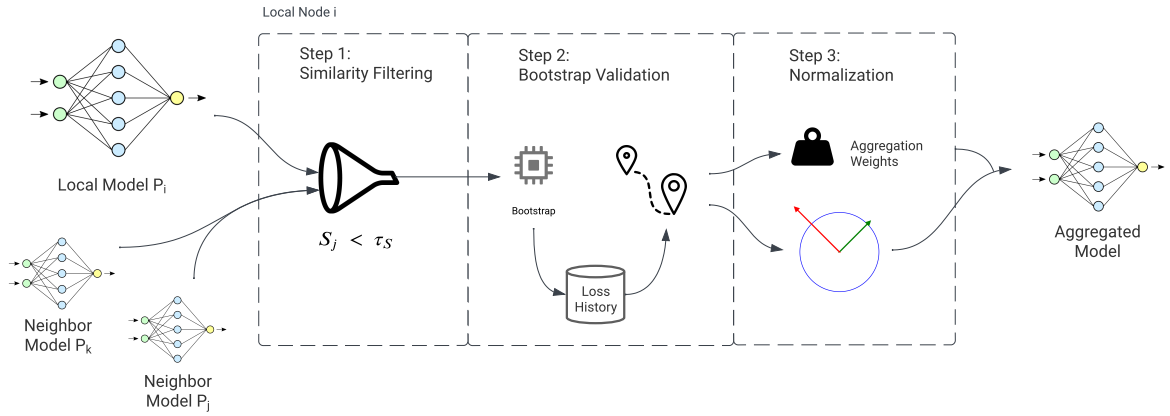


Figure 4.2: High-level overview of the aggregation process in Sentinel.

(1) Similarity Filtering. At the first stage, Sentinel computes the layer-wise average cosine similarity between the current local model and all neighbor models received. This similarity is denoted as S_j. The computation of the cosine similarity is defined in Algorithm 2. The cosine similarity is chosen due to its advantage over the Euclidean distance, as discussed in Section 3.1. In contrast to existing methods, Sentinel does not cluster or weigh models based on cosine similarity, but rather filters them instead. This step therefore serves as a preliminary similarity evaluation to exclude potentially malicious models

Algorithm 1 Sentinel Aggregation Algorithm

Require: P: Neighbor parameters, D_{bs}: Bootstrap dataset, H: Loss history, τ_S: Similarity threshold, τ_L: Loss distance threshold.

$r \leftarrow$ current round
$M \leftarrow$ local model
$w \leftarrow 0$
for j in P, $j \neq i$ **do** ▷ **(1) Similarity filtering**
    $S_j \leftarrow CosineSimilarity(P_j, M)$
    if $S_j < \tau_S$ **then** Remove P_j from P
$H_i[r] \leftarrow ComputeBootstrapLoss(M, D_{bs})$ ▷ **(2) Bootstrap validation**
for j in P, $j \neq i$ **do**
    $l_j \leftarrow ComputeBootstrapLoss(P_i, D_{bs})$
    $H_j[r] \leftarrow l_j$ ▷ Update loss history
    $w_j \leftarrow MapLossDistance(H_i, H_j, \tau_L)$
for j in N, $j \neq i$ **do** ▷ **(3) Normalization**
    $\tilde{P}_j \leftarrow NormaliseModel(P_i, P_j)$
$\theta \leftarrow \frac{1}{\sum_{j \in N} w_j} \left(\sum_{j \in N} w_j \tilde{P}_j \right)$ ▷ Aggregate
return θ

Algorithm 2 CosineSimilarity

Require: P: Neighbor parameters, M: Local model parameters
Ensure: $|M| = |P|$

for l_m, l_p in M, P **do**
    $S_P \leftarrow S_P + \phi(\frac{\mathbf{v_1} \cdot \mathbf{v_2}}{\|\mathbf{v_1}\| \|\mathbf{v_2}\|})$ ▷ row-wise average
$S_P \leftarrow S_P / |M|$ ▷ layer-wise average
return S_P

and reduce the computational overhead of the second stage. The decision boundary is defined by τ_S, such that all models exhibiting a lower cosine similarity to the local model than τ_S will be filtered.

(2) Bootstrap Validation. At the second stage, the remaining models are evaluated on the basis of their loss. To do so, each node holds a small bootstrap dataset D_{bs}, which is randomly sampled from the validation dataset of the same node. The size of this bootstrap dataset is set to a third of the validation dataset or at least 300 samples, *i.e.*, $|D_{bs}| = max(|D_{val}|, 300)$. The number of collected samples required to effectively measure the loss has been investigated in other works and is therefore not within the scope of this thesis [32], [46], [49]. In general, choosing the bootstrap dataset size is a trade-off between computational overhead and performance. Therefore, the chosen size should be suitable for the device specifications of each participating node. The computation of the bootstrap validation loss is defined in Algorithm 3. Note that the local node also computes the bootstrap validation loss for its own local model for a more accurate comparison.
After computing the performance of each received model, Sentinel computes the average of the loss history up to the current aggregation round. This procedure simply takes all previously computed loss values into account. Thus, if the bootstrap loss was not

computed in a previous round, since the model was filtered in step (1), there will be consequently fewer values to be averaged. Generally, the averaging serves for better computational stability. Finally, all average loss values are then compared to the local average bootstrap loss according to Algorithm 4, which results in an aggregation weight w_i.

Sentinel maps the average loss distance according to a damped decay function, illustrated in Figure 4.3. The damping factor κ is defined as the inverse of the average local loss. In theory, the local loss could be 0. Hence, a minimum average loss l_{min} must be defined for numerical stability, *e.g.*, $l_{min} = 0.001$. Consequently, the local model and any neighbor model that presents a loss lower than the local model will receive $w = 1$. With this approach, Sentinel becomes more defensive as the average local loss decreases, which is expected during the FL process. If this does not occur and the local loss increases, *i.e.*, the node is not learning, and Sentinel becomes more exploratory.

Algorithm 3 ComputeBootstrapLoss

Require: P: Model parameters, D_{bs}: Bootstrap dataset

1: **for** each (x, y) in D_{bs} **do**
2: $\quad y_{pred} \leftarrow$ Predict with P on x
3: $\quad L \leftarrow \frac{1}{|D_{\text{bs}}|} \sum_{i=1}^{|D_{\text{bs}}|} l\left(y, y_{\text{pred}}\right)$ ▷ Compute loss
4: **return** L

Algorithm 4 MapLossDistance

Require: H_i: Local loss history, H_l: Neighbor loss history, τ_L: Loss distance threshold.

1: $\bar{l_i} \leftarrow \phi(H_i)$, $\bar{l_j} \leftarrow \phi(H_j)$ ▷ Average current loss history
2: $\kappa \leftarrow max(\bar{l_i}, l_{min})^{-1}$ ▷ Damping factor
3: $\bar{d_l} \leftarrow max(\bar{l_j} - \bar{l_i}, 0)$
4: $w \leftarrow \exp\left(-k * \bar{d_l}\right)$
5: **if** $w < \tau_L$ **then** $w = 0$
6: **return** w

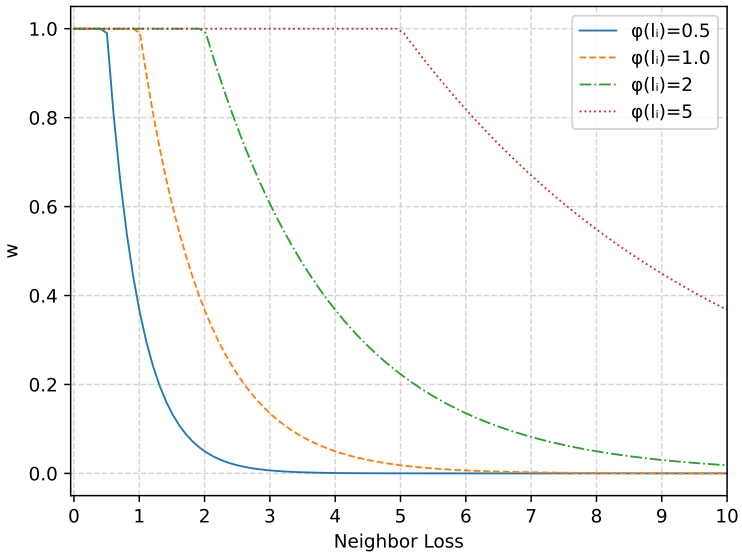


Figure 4.3: An illustration of the *MapLossDistance* procedure for different average local bootstrap loss values $\phi(l_i)$. The threshold τ_L is not displayed.

(3) Layer Normalization. Inspired by recent normalization approaches [24], [32], [52], the last step of Sentinel is the normalization of models to defend against potential stealth attacks that were able to pass defense layers (1) and (2). This procedure is defined in Algorithm 5. The normalization reduces the magnitude of potentially scaled attacks, which are commonly used to introduce backdoors. Such that Sentinel does not rely on a threshold, the ratio of the local model norm and the neighbor model norm is used as a scaling factor ρ. Sentinel only decreases the norm of neighbor models, hence $\rho \leq 1$. Subsequently, the normalized models are aggregated according to Algorithm 1 line 14.

Algorithm 5 NormalizeModel

Require: P: Neighbor parameters, M: local model parameters

1: $L \leftarrow$ number of layers
2: **for** l in $|M|$ **do**
3: $\quad \rho \leftarrow min(1, \frac{\|P[l]\|}{\|M[l]\|})$
4: $\quad \tilde{P}[l] \leftarrow \rho * P[l]$
5: **return** $\tilde{P}$

4.2.2 SentinelGlobal

As an extension of Sentinel, a global trust variant is proposed that aims at reducing the computational complexity of the individual defense stages. This extension is specified therein as SentinelGlobal. The fundamental idea of SentinelGlobal is to expand the model evaluation to a cluster of trustworthy nodes. Thereby, a global network perspective on potentially malicious actors is obtained. A single node may not be able to reliably detect a poisoned model due to the heterogeneity of the local data. Combining the neighboring opinions creates a collaborative decision that overcomes this limitation.

With the first step of cosine filtering and the second step of bootstrap validation, node n_i applying Sentinel either aggregates a neighbor model P_j to some extent, or excludes the update entirely from the aggregation. This decision can be mapped to a binary trust value $\mathcal{X}$, where $\mathcal{X} = 0$, if the model is excluded from aggregation (either by cosine similarity or bootstrap validation), and $\mathcal{X} = 1$ if the model is included in the aggregation. Consequently, a local node n_i evaluating parameters from node j at round r indirectly computes a trust perspective $\mathcal{X}_{ij}^r \in \{0, 1\}$. The X_i^r is thus the set of trust perspectives on each neighbor of node n_i. Note that a node always trusts itself, *i.e.*, $\mathcal{X}_{ii} = 1$. The local trust scores can then be shared with the neighboring nodes to create a global trust perspective. This concept is incorporated into SentinelGlobal, defined in Algorithm 6, which is based on Sentinel as a baseline aggregation algorithm.

SentinelGlobal becomes active after a predefined activation round r_α. The mechanisms can only be activated by the second round earliest, since the trust scores of the previous round are required. Every node computes its local trust scores at each round, independently of r_α, and sends this perspective to its neighbors. Once $r \leq r_\alpha$ holds true, the procedure *TrustedNeighborOpinion* (Algorithm 7) decides whether a neighbor model P_j should be evaluated by Sentinel or discarded ahead of time. For a target node j, this procedure collects the trusted neighbors of the node n_i, *i.e.*, $\mathcal{X}_{ik}^r = 1, k \in N$ and averages all trusted opinions on the target node n_j. The average trusted neighbor opinion $\tilde{\Phi}_j$ is then compared to the trust threshold τ_χ.

Figure 4.4 illustrates an example, where five nodes represent a fully connected network. At the current round $r + 1$, node n_1 evaluates the global trust of n_2. From the local trust of the previous round $\mathcal{X}_1^r$, the trusted neighbors are n_3 and n_4. The average trusted neighbor opinion is then computed as follows: $\tilde{\Phi}_j \leftarrow \phi(\mathcal{X}_{12}^r, \mathcal{X}_{32}^r, \mathcal{X}_{42}^r) = \phi(0, 1, 0) = \frac{1}{3}$.

Algorithm 6 SentinelGlobal Aggregation Algorithm

Require: P: Neighbor parameters, D_{bs}: Bootstrap dataset, H: Loss history, τ_S: Similarity threshold, τ_χ: Global trust threshold. r_α: Activation round τ_L: Loss distance threshold.

$r \leftarrow$ current round
$M \leftarrow$ local model
$w \leftarrow 0$
for j in N, $j \neq i$, $r \geq r_\alpha$ **do** ▷ **(0) Trust evaluation**
    $\tilde{\Phi}_j \leftarrow TrustedNeighbourOpinion(\mathcal{X}, j)$
    if $\tilde{\Phi}_j < \tau_\chi$ **then** Remove P_j
for j in P, $j \neq i$ **do** ▷ **(1) Cosine filtering**
    $S_j \leftarrow CosineSimilarity(P_j, M)$
    if $S_j < \tau_S$ **then** Remove P_j from P
$H_i[r] \leftarrow ComputeBootstrapLoss(M, D_{bs})$ ▷ **(2) Bootstrap validation**
for j in P, $j \neq i$ **do**
    $l_j \leftarrow ComputeBootstrapLoss(P_i, D_{bs})$
    $H_j[r] \leftarrow l_j$ ▷ Update loss history
    $w_j \leftarrow MapLossDistance(H_i, H_j, \tau_L)$
for j in P, $j \neq i$ **do** ▷ **(3) Normalization**
    $\tilde{P}_j \leftarrow NormaliseModel(P_i, P_j)$
$\theta \leftarrow \frac{1}{\sum_{j \in N} w_j} \left(\sum_{j \in N} w_j \tilde{P}_j \right)$ ▷ Aggregate
$\mathcal{X}_i^r \leftarrow P$ ▷ Update local trust
Send $\mathcal{X}^r$ to neighbors
$\mathcal{X}_j^r \leftarrow$ Receive neighbor trust from S_j
return θ

Algorithm 7 TrustedNeighborOpinion

Require: X: Trust, j: Target index

$\Phi \leftarrow 0$
$n_t \leftarrow 0$
for k in N **do**
    if $\mathcal{X}_{ik}^{r-1} = 1$ **then** ▷ Trusted neighbors
        $\Phi \leftarrow \Phi + \mathcal{X}_{kj}^{r-1}$ ▷ Add Neighbor Opinion
        $n_t = n_t + 1$
$\tilde{\Phi} \leftarrow \Phi / n_t$
return $\tilde{\Phi}$

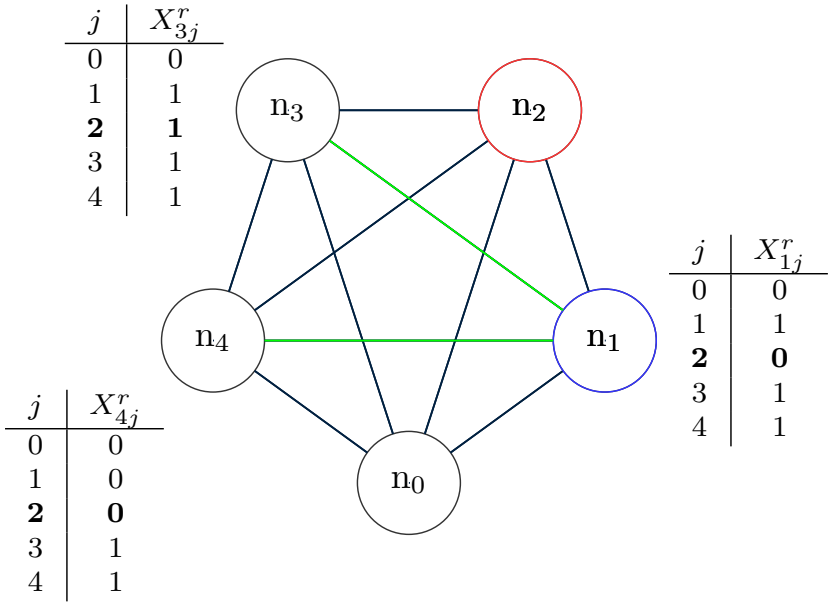


Figure 4.4: An example of the *TrustedNeighborOpinion* for node n_1 (blue) and target node n_2 (red). Only required $\mathcal{X}$ shown due to spacing. Node n_1 trusts n_3 and n_4 (green), thus considers their trust opinion on n_2.

Chapter 5

Implementation

Fedstellar [55] was chosen as the underlying platform to run DFL scenarios. Fedstellar, implemented in Python, is designed to configure and run FL scenarios with a wide range of customization options. It is available open source on GitHub[1]. The platform allows the user to execute the FL process in a decentralized, semi-decentralized, and centralized manner. It deploys the desired scenario to each specified client and manages the network of participants in terms of node connectivity. The user can choose to run the FL scenario on selected physical devices or in a containerized simulation on Docker[2]. It is fully extensible, such that the user can implement custom models, load new datasets, and define the preferred aggregation algorithms. Furthermore, Fedstellar is ML framework-agnostic. In the context of this work, PyTorch Lightning[3] was used to perform deep learning. In the following, the required adaptions to Fedstellar for this thesis and the implementation details of Sentinel and SentinelGlobal are discussed.

5.1 Adaptions to Fedstellar

Fedstellar defines three types of roles each node can take up: aggregators, which train and aggregate models; trainers, which only train and distribute models, but do not aggregate; and proxies, which simply act as a network hop for model distribution. For simplicity, this work only considers aggregators. Additionally, Fedstellar executes the FL process in asynchronous mode (at the time of this thesis). That is, each node is allowed to update its local model independently and asynchronously, without waiting for other nodes to finish their updates. There are certain advantages over a synchronous FL process, such as robustness to node failures. However, due to this asynchronous constellation, a node is not guaranteed to aggregate a specified number of models at each round, which yields some non-determinism in the FL process. To make experiments more reproducible, an adapted, synchronous version has been implemented. This fork of Fedstellar, further referred to as

[1] https://github.com/enriquetomasmb/fedstellar
[2] https://www.docker.com/
[3] https://www.pytorchlightning.ai

Fedstellar-Sync, is also available on GitHub[4].
In the following, the changes required to arrive at Fedstellar-Sync are discussed. Beforehand, the original core procedure of Fedstellar executed by each node is clarified on a high level. The corresponding definition can be retrieved from Algorithm 8. At the start of the FL scenario, the controller[5] preliminary defines a starter node. Each node is individually responsible to define or acquire the local training, validation and test dataset. This selected starter node then commands all nodes to start learning, including itself. Each node owns an aggregator instance, which is responsible for collecting models and performing the aggregation. While a node is training, it can receive other models and add them to its aggregator. Once the training finishes, the node's aggregator performs a partial aggregation of all models received up to this point. A gossiping algorithm then distributes this partial aggregation repeatedly. Details about the gossiping algorithm can be retrieved from the code of Fedstellar[6]. On the other hand, if a node takes too long to train and the aggregator reaches a timeout, the node stops its local training and continues the partial aggregation without its own model. Subsequently, the node proceeds with the next round, independently of the state of other nodes. After completing R rounds, the node tests its final model on the local test dataset and completes its FL process.

For the synchronous version of Fedstellar, a number of adaptions have been implemented. An abstract definition is outlined in Algorithm 9. First, before any node starts its initial round, the connection to all neighbors is ensured. This change is implemented in `node_start.py`[7]. Additionally, the node connection timeout in `node_connection.py`[8] has been removed. Secondly, once the local training is finished, only the local model is broadcasted to the corresponding neighbors. The gossiping and partial aggregation have been removed. Instead of an aggregation timeout, the aggregator of each node waits until it has received the local model and a model from every neighbor. Consequently, this adjustment ensures that all nodes terminate their local training round, followed by a complete aggregation. The technical specifications are available in the abstract aggregator class `aggregator.Aggregator`[9]. As a final adaption, a synchronization barrier ensures that all nodes complete the current round before proceeding to the next. This is formally defined in Algorithm 9 line 15 onwards. Details can be retrieved from the code in `node.Node.__on_round_finished`[10]. With these changes in place, every node is ensured to aggregate each neighbor model and synchronize the round progression. Correspondingly, if a one or more nodes fail, the FL scenario is terminated holistically.

[4]https://github.com/janousy/fedstellar-sync
[5]The controller represents the administrative instance of Fedstellar and orchestrates the scenario setup.
[6]fedstellar/gossiper.py
[7]fedstellar/node_start.py
[8]fedstellar/node_connection.py
[9]fedstellar/learning/aggregators/aggregator.py
[10]fedstellar/node.py

Algorithm 8 Federated participant cycle in Fedstellar, adapted from [55]

Require: R: local round, α: learning rate, λ: regularization parameter, S_j: socket to neighbor j, D: local dataset

$D_{\text{Train}}, D_{\text{Train}} \leftarrow \text{split}(D)$
for r in R **do**
  Initialize Local Model with Parameters θ
  for each (x, y) in D_{Train} **do**
    $\theta \leftarrow \theta - \alpha(\nabla J\theta, x, y) + \lambda\theta$ ▷ Train
  while not agg_timeout **do** } in parallel
    for j in N **do**
      $P_j \leftarrow$ add model parameters from j via S_j ▷ Receive
  $\theta \leftarrow GossipModelAggregation(P)$ ▷ Aggregate
  Update Local Model with θ
for each (x, y) in D_{Test} **do**
  $y_{pred} \leftarrow$ Predict with Local Model on x ▷ Test
  $L \leftarrow \frac{1}{|D_{\text{Test}}|} \sum_{i=1}^{|D_{\text{Test}}|} l\left(y, y_{\text{pred}}\right)$ ▷ Compute Loss

procedure GossipModelAggregation(P)
  $P_{agg} = Aggregator.getPartialAggregation(P_{curr})$
  for j in N **do**
    Send $P_{agg} \leftarrow S_j$ ▷ Send

Algorithm 9 Synchronous participant cycle in Fedstellar

Require: R: local round, α: learning rate, λ: regularization parameter, S_j: socket to neighbor j, D: local dataset

$D_{\text{Train}}, D_{\text{Train}} \leftarrow \text{split}(D)$
for j in N **do** ▷ Connect
  connect S_j
for r in R **do**
  Initialize Local Model with Parameters θ
  for each (x, y) in D_{Train} **do**
    $\theta \leftarrow \theta - \alpha(\nabla J\theta, x, y) + \lambda\theta$ ▷ Train
  Broadcast P_i after training
  while not all models received **do** } in parallel
    for j in N **do**
      $P_j \leftarrow$ add model parameters from j via S_j ▷ Receive
  $\theta \leftarrow Aggregator.aggregate(P)$ ▷ Aggregate
  Update Local Model with θ
  for j in N **do**
    Send *model_ready*
  while not all *model_ready* **do**
    wait() ▷ Sync Barrier
for each (x, y) in D_{Test} **do**
  $y_{pred} \leftarrow$ Predict with Local Model on x ▷ Test
  $L \leftarrow \frac{1}{|D_{\text{Test}}|} \sum_{i=1}^{|D_{\text{Test}}|} l\left(y, y_{\text{pred}}\right)$ ▷ Compute Loss

5.2 Aggregation

Fedstellar follows a composite design, where each node initializes a predefined type of aggregator at the setup of the FL scenario. An aggregator represents an abstract class that manages the aggregation of models, as concisely defined in Listing 1. More technical details can be retrieved from `aggregator.py`[11]. By default, this parent class implements every member function, except `aggregate()`. Consequently, specific aggregation algorithms extend this class and are required to implement the `aggregate()` function.

Generally, the aggregator is a thread that is started by adding the first model at each round. Thereby, the aggregation will be executed in the background as soon as all models are added. Moreover, this thread represents an observable. Hence, the node registers as an observer to become notified once the aggregation has completed. The `add_model()` function in Fedstellar-Sync was extended to accept a general `ModelMetrics` object instead of just the number of samples. This dictionary can compromise custom metrics such as cosine similarity, bootstrap validation loss or trust scores. The function `check_and_run_aggregation` is called as soon as the last model is added to the aggregator, which then ensures that all models are available, and finally executes the aggregation.

Listing 1 Aggregator Parent Class

```
class Aggregator(threading.Thread, Observable):
    def run(self):
        # Wait for the aggregation. Then, notify node.
    def aggregate(self, models):
        # Aggregate the models with custom algorithm.
    def add_model(self, model: OrderedDict, nodes: List[str], metrics: ModelMetrics):
        # Add a model. The first model to be added starts the `run` method.
    def get_local_model(self):
        # Retrieve the local model from the aggregator.
    def get_full_aggregation(self):
        # Aggregate all current models.
    def check_and_run_aggregation(self):
        # Check if all models have been added and start aggregation if so.
    def clear(self):
        # Clear all for a new aggregation.
```

[11] fedstellar/learning/aggregators/aggregator.py

5.2.1 Sentinel

The definition of Sentinel given in Algorithm 1 was implemented for Fedstellar in Python. Listing 2 clarifies how the aggregation function of the parent class was overwritten. The individual functions used in the aggregation (`filter_models_by_cosine()`, `get_mapped_avg_loss()`, `normalise_layers()`) were implemented according to Algorithms 1, 4 and 5, respectively. The procedure `evaluate_model()` computes the cosine similarity and bootstrap validation loss (for sufficiently high S_c). Further technical specifications can be retrieved from the file `sentinel.py`[12]. Note that in the specific case of Fedstellar, Sentinel does not aggregate in round 0, as this round is a diffusion round to synchronize the model initialization on each node. Instead, Sentinel returns the local model at round 0. Technically, it would be possible to evaluate a model within `add_model()` while a node is still training. However, this would result in a loss of generality and impact the comparability to other algorithms.

5.2.2 SentinelGlobal

The global trust extension to Sentinel was implemented according to Algorithm 6. An extract of relevant functions is available in Listing 3. The trust scores are represented as dictionaries, with a key value pair for every node. When a model is added to SentinelGlobal via `add_model()`, the trust perspective of the corresponding neighbor is extracted and added to the local dictionary, which keeps track of all neighbor opinions for each round. The aggregate procedure is implemented similar to Sentinel, but adds an additional step 0 for the trust evaluation. Therefore, the function `evaluate_model_trusted` computes the trusted neighbor opinion as defined in Algorithm 7. If the neighbor is not trusted, the model metrics are set accordingly, such that the model becomes excluded in step 1 of similarity filtering. Otherwise, the model is evaluated. Finally, the current local trust scores, resulting from the cosine filtering and loss distance mapping, are added to the node's trust records for the next round. Further details are available in the file `sentinelglobal.py`[13].

[12] fedstellar/learning/aggregators/sentinel.py
[13] fedstellar/learning/aggregators/sentinelglobal.py

Listing 2 Sentinel (logging removed)

```python
class Sentinel(Aggregator):
    # [...]

    def aggregate(self, models):
        # Compute metrics
        local_model = models.get(self.node_name)
        for node_key in models.keys():
            model = models[node_key][0]
            metrics: ModelMetrics = models[node_key][1]
            metrics_eval = self.evaluate_model(model, node_key, metrics)
            models[node_key] = (model, metrics_eval)

        # Step 1: Evaluate cosine similarity
        filtered_models = filter_models_by_cosine(models, self.similarity_threshold)
        if len(filtered_models) == 0:
            return models.get(self.node_name)[0]

        # Step 2: Evaluate bootstrap validation loss
        loss = {}; mapped_loss = {}; cos = {}
        for node_key, msg in filtered_models.items():
            params = msg[0]
            metrics: ModelMetrics = msg[1]
            loss[node_key] = metrics.validation_loss
            mapped_loss[node_key] = self.get_mapped_avg_loss(node_key,
            ↪  metrics.validation_loss)
            cos[node_key] = metrics.cosine_similarity

        # Step 3: Normalise the remaining (filtered) untrusted models
        normalised_models = {}
        for key, msg in filtered_models.items():
            model_params = msg[0]
            metrics = msg[1]
            if key == self.node_name:
                normalised_models[key] = (local_params, metrics)
            else:
                normalized_params = normalise_layers(model_params, local_params)
                normalised_models[key] = (normalized_params, metrics)

        # Aggregate
        accum = (list(normalised_models.values())[-1][0]).copy()
        for layer in accum:
            accum[layer] = torch.zeros_like(accum[layer])
        for node, message in normalised_models.items():
            model = message[0]
            weight = mapped_loss[node] / sum(mapped_loss.values())
            for layer in model:
                accum[layer] = accum[layer] + model[layer] * weight
        return accum

```

Listing 3 SentinelGlobal Implementation (logging removed)

```python
class SentinelGlobal(Aggregator):
    # [...]

    def aggregate(self, models):
        # Step 0: Compute trust, metrics if trusted
        for node_key in models.keys():
            model = models[node_key][0]
            metrics: ModelMetrics = models[node_key][1]
            # the own local model also requires eval to get loss distance
            metrics_eval = self.evaluated_model_trusted(model, node_key, metrics)
            models[node_key] = (model, metrics_eval)

        # [...] Sentinel.aggregate

        # Store local trust scores
        for node_key in models.keys():
            if node_key in malicious and node_key != self.node_name:
                self.global_trust[self.agg_round][self.node_name][node_key] = 0
            else:
                self.global_trust[self.agg_round][self.node_name][node_key] = 1
        return accum

    def evaluate_model_trusted(self, model: OrderedDict, node: str, metrics:
    ↪ ModelMetrics):
        # Evaluate if not yet active
        if self.agg_round < self.active_round:
            metrics = self.evaluate_model(model, node, metrics)
            super().add_model(model=model, nodes=[node], metrics=metrics)
            return

        # Step 0: Check whether the model should be evaluated based on global trust
        avg_global_trust = self.get_trusted_neighbour_opinion(node)
        if avg_global_trust < TRUST_THRESHOLD and node != self.node_name:
            metrics.cosine_similarity = 0  # model will be removed by similarity
            ↪ filtering
            metrics.validation_loss = float('inf')
        else:
            metrics = self.evaluate_model(model, node, metrics)

        super().add_model(model=model, nodes=[node], metrics=metrics)
        return metrics

    def add_model(self, model: OrderedDict, nodes: List[str], metrics: ModelMetrics):
        for node in nodes:
            self.neighbor_keys.add(node)
        # Add the received trust metrics to the local trust record
        self.add_neighbour_trust(metrics.global_trust)
        super().add_model(model=model, nodes=nodes, metrics=metrics)

```

Chapter 6

Evaluation

Hereinafter, an evaluation of the two defense mechanisms Sentinel and SentinelGlobal will be provided, in comparison to other state-of-the-art aggregation algorithms on various datasets. This chapter will first outline the experimental setup, then present the results of the experiments, and conclude with a discussion of the observations.

6.1 Experiment Setup

The following sections will specify the datasets, their corresponding model and a selection of reference algorithms that were used to evaluate Sentinel and SentinelGlobal. Furthermore, the threat model and the configuration of Fedstellar-Sync will be discussed.

6.1.1 Datasets and Models

For this work, three datasets and a corresponding deep learning model were chosen to evaluate the aggregation algorithms implemented for Fedstellar-Sync. The distribution of the following datasets is considered IID:

- **MNIST** [56] is a popular baseline benchmark for neural networks. It consists of handwritten digits represented by grayscale level pixels of size 28×28. It compromises 60 000 training samples and 10 000 test samples. The labels are represented by 10 classes, which correspond to the digits from 0 to 9.
  The chosen model[1] to learn the MNIST dataset is a multilayer perceptron (MLP) with a linear input layer of size $28 * 28 \times 256$, a linear hidden layer of size 256×128 and a linear output layer of size 128×10. The input layer and the hidden layer use a ReLU activation function, the output layer applies softmax to produce the final

[1] fedstellar/learning/pytorch/emnist/models/mlp.py

logits. The optimizer and the learning rate are set to Adam[2] and $1e{-}3$, respectively. As a loss function, the cross-entropy loss is applied. This MLP is trained for 3 epochs per federated round.

- **FashionMNIST** (FMNIST) [57] was created as a drop-in replacement for MNIST to increase the difficulty of the learning task. Similarly, it consists of 60 000 training samples and 10 000 test samples, which are 28×28 grayscale images. Instead of digits, the 10 labels represent clothing articles such as "trouser", "pullover" or "dress". For the FMNIST dataset, the same MLP as for MNIST is used. Again, the model is trained for 3 epochs per round.

- **CIFAR10** [58] is a widely used dataset in computer vision as a benchmark for object recognition. It consists of 10 classes corresponding to well-defined objects such as "airplane", "automobile", or "bird". The samples are represented as 32×32 color images. In total, there are 50 000 training and 10 000 test images.
  To train an object recognition model on the CIFAR10 dataset, a small convolutional neural network (CNN) designed for mobile applications is used [59]. The model architecture[3] is briefly summarized as follows: the initial input layer of the model consists of a standard convolutional layer with 3 input channels, 32 output channels, a kernel size of 3, a stride of 1, and padding of 1. This is followed by batch normalization and ReLU activation. Subsequently, the input layer is fed into 5 depthwise separable convolutional layers with different input and output channel sizes and strides. These layers help in reducing the computational cost of the model while preserving expressive power. Finally, global pooling in the form of an adaptive average pooling layer is applied to reduce the spatial dimensions of the output feature map to a size of 1×1. This is followed by a fully connected layer that maps the output of the previous layer to the required number of output classes. The optimizer and the learning rate are set to Adam and $1e{-}3$, respectively. This convolutional network is trained for 5 epochs per round.

6.1.2 Selected Reference Algorithms

Such that the defense techniques in this work can be evaluated, three state-of-the-art defense techniques are chosen as a reference for effectiveness: Krum, TrimmedMean, FlTrust. These aggregation algorithms are selected based on their performance against various poisoning attacks and their transferability to DFL. Additionally, FedAvg serves as a baseline for the FL process under benign assumptions. In the following, the approach and the configuration of all aggregators investigated are briefly described.

- FedAvg (Section 2.1.1) averages all parameters layer-wise, weighted by the number of samples used to train the respective model.

- TrimmedMean (Section 3.1) computes the aggregated model with a coordinate-wise mean, but removes the largest and smallest β of them, and computes the mean of

[2] https://pytorch.org/docs/stable/generated/torch.optim.Adam.html
[3] fedstellar/learning/pytorch/cifar10/models/simplemobilenet.py

the remaining $m - 2\beta$ parameters for each layer. For the experiments of this work, TrimmedMean is configured with a trimming parameter of $\beta = 1$.

- Krum (Section 3.1) selects one of the m local models that is most similar to all other models as the next local model based on Euclidean distance. Krum does not accept any configuration parameters.

- FlTrust (Section 3.3) represents an adapted version for DFL. Instead of using a root dataset to train a reference model, all received neighbor models are compared to the locally trained model in terms of the layerwise, ReLu-clipped cosine similarity, which serves as a trust score. Every model is normalized using the norm of the local model, and then aggregated based on the respective trust score.

- Sentinel (Section 4.2) aggregates according to Algorithm 1. For the experiments in this work, Sentinel is configured with $\tau_{cos} = 0.5$ and $\tau_{loss} = 0.5$.

- SentinelGlobal (Section 4.2) aggregates according to Algorithm 6. SentinelGlobal is configured with $\tau_{cos} = 0.5$ and $\tau_{loss} = 0.5$, and additionally $\tau_{\chi} = 0.5$. Thereby, the majority of trusted nodes need to agree on an evaluation of a target node. SentinelGlobal becomes active after $r_{\alpha} = 3$ for all three datasets: MNIST, FMNIST and CIFAR10.

6.1.3 Threat Model

As poisoning attacks in FL are the focus of this work, all aggregators mentioned in Section 6.1.2 were tested against various poisoning attacks and different configurations. The following attacks have been selected for evaluation: targeted and untargeted label flipping, artificial backdoor attack (targeted sample poisoning) and untargeted model poisoning. Technical details can be retrieved from Section 4.1. The targeted label flipping attack always tries to provoke a misclassification from samples with label l_3 to label l_7, independent of the dataset. For MNIST, this targets the digit 4 to be misclassified as an 8. For FMNIST, this involves the images illustrating a "dress" to be classified as a "bag". And finally, for CIFAR10, label l_3 is represented by "cat" and l_7 represents "horse". Similarly, the backdoor is configured to the target label l_3 for all datasets.

As of the parameters, the PNR was set to 10, 50, and 80%, *i.e.*, a low, medium, highly poisoning environment. Similar to the PNR, the attackers executed a light, medium, and strong attack with a PSR of 30, 50, and 100% where applicable. For the model poisoning, a salt NR of 80% was chosen. Lower salt NR are not effective enough to degrade the model performance, even for FedAvg. The attack parameters are summarized in Table 6.1. Per aggregator, this resulted in 3 experiments for model poisoning and 9 experiments for each other attack. Per dataset, 60 experiments have been conducted, and thus 180 experiments in total for all datasets.

Table 6.1: Configuration overview for each of the selected attacks. Cells marked with "-" specify that the parameter is not applicable to the corresponding attack.

Attack Type	**Parameter**			**Metric**
	PNR [%]	PSR [%]	NR [%]	
Model Poisoning	(10, 50, 80)	-	80	F1-Score
Untargeted Label Flipping	(10, 50, 80)	(30, 50, 100)	-	F1-Score
Targeted Label Flipping	(10, 50, 80)	(30, 50, 100)	-	ASR_{lf}
Backdoor	(10, 50, 80)	(30, 50, 100)	-	BA

6.1.4 Fedstellar Configuration

In the following paragraphs, the internal configuration of Fedstellar is described. Although the adapted version Fedstellar-Sync was used for all experiments, the configurations listed hereinafter are also applicable to the base version. Hence, this section does not further distinguish between these two versions.

For the network constellation, a total number of nodes $|N| = 10$ was used. The network has been configured to be fully connected, *i.e.*, each node was assigned to 9 neighbors, as illustrated in Figure 6.1. The set of neighbors remained constant, *i.e.*, node connections were kept established. The position of malicious nodes was assigned randomly. Note that with a maximum PNR of 80%, defined in Section 6.1.3, and 10 nodes, there are always at least 2 benign participants. Each participating node took up the role of an aggregator, *i.e.*, there were no proxy nodes. Note that an aggregator in DFL also trains its own model.

As for the aggregation algorithm in place, all nodes applied the same aggregation technique, whether malicious or benign. Furthermore, the data was split equally among all nodes with an IID distribution. In the case of MNIST and FMNIST, each node consequently received 6 000 training samples and 1 000 test samples. For CIFAR10, there were 5 000 training samples and 1 000 test samples locally available at each node. The splitting factor to obtain a validation dataset from the training samples was set to 10% for all experiments.

The duration of the FL process has been configured to 10 rounds for all experiments. It is worth mentioning that Fedstellar executes a model diffusion in round 0, such that there are actually 11 rounds in total, but the local training occurs only for the remaining 10 rounds. This number of rounds is not necessarily required to achieve an acceptable model performance, but it gives the attackers enough time to poison the models of benign participants. The network arguments of Fedstellar were set to be efficient, *i.e.*, the rate was configured to 1 Mbps, the loss to 0% and the delay to 0 ms. Thus, the results of the experiments were not influenced by inconsistent network behavior. A full list of Fedstellar specific configuration parameters can be retrieved from Appendix B.1.
All experiments were simulated using the Docker functionality of Fedstellar. In terms of hardware, two different single machine environments were used as an experiment platform.

A CPU environment was used for MNIST and FMNIST, and a GPU environment for CIFAR10:

- CPU environment: AMD EPYC 7502P 32-Core Processor @ 2.50GHz, 48 Sockets, 3-level cache hierarchy and AVX2 support, 100 GB RAM, OS: Ubuntu.
- GPU environment: (2) NVIDIA GeForce RTX 3080, Intel(R) Core(TM) i7-10700F CPU @ 2.90GHz, 16 Sockets, 3-level cache hierarchy and AVX2 support, 100 GB RAM, OS: Ubuntu.

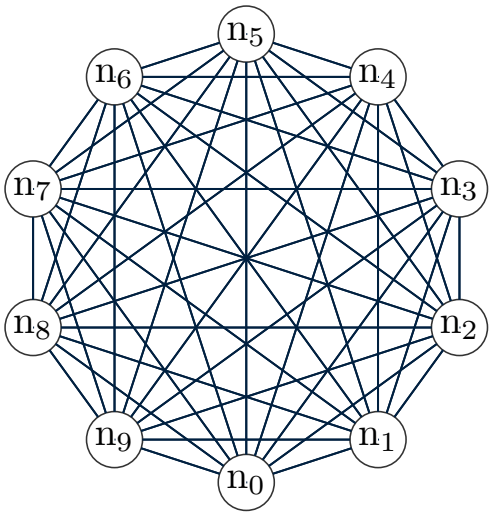


Figure 6.1: The Fedstellar network constellation used for all experiments, 10 nodes in a fully connected network.

6.2 Results

In the following, the evaluation of the performance of Sentinel and SentinelGlobal in comparison to other protocols is reported. First, a baseline performance reference under benign settings is established. Subsequently, each attack is discussed individually for all datasets. If not otherwise stated, the metrics were evaluated on the test dataset of each local node. Where applicable, the Standard Error Mean (SEM) is reported. A collection of the raw result data is available in the GitHub repository of Fedstellar-Sync[4].

6.2.1 Baseline Performance

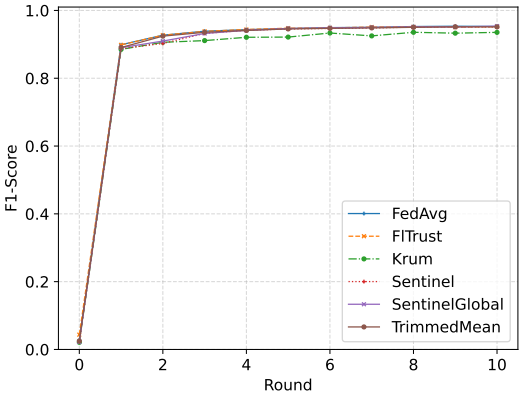
The baseline performance serves as a reference for subsequent experiments in a malicious environment, such that the effectiveness of a defense mechanism can be evaluated. Additionally, it is important to measure the baseline performance of an aggregation algorithm to prevent any interference with the FL process under normal circumstances.

[4]evaluation/results

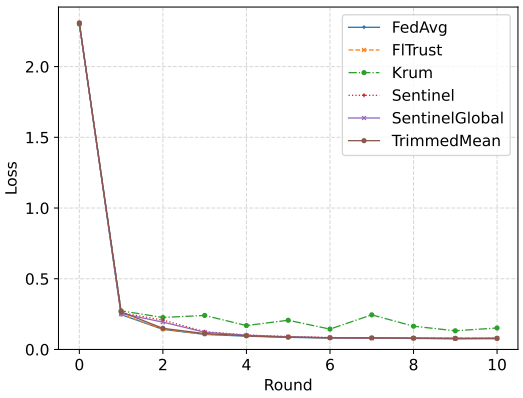
Table 6.2 gives an overview of the behavior of the selected aggregation algorithms in a benign environment, *i.e.*, there are no poisoning participants. The metrics were calculated using the average metric of all participants after completing the final round 10. For the F1-Score, SentinelGlobal performed the best for MNIST, whereas FedAvg reached the highest F1-score for FMNIST and CIFAR10. For all datasets, the SEM F1-score lied between 0.036 and 0.093. Therefore, there was a negligible variability in terms of performance between participants. This can be explained by the stochastic nature of machine learning in general.

In particular, the average SEM F1-score positively correlated with the average F1-score. On MNIST, where the highest F1-scores were reached, the SEM was almost twice as high as the SEM for CIFAR10, where the lowest F1-scores were reached. This also holds true for the average loss values. In particular, Krum performed overall the worst in the benign environment. For all datasets, the aggregator reached the lowest F1-score and the highest loss. The performance drop increased with task complexity, *i.e.*, for CIFAR10, Krum exhibited a F1-score difference of approximately 0.1 compared to FedAvg. This is likely due to the single model selection approach of Krum, whereby no model knowledge can be combined. In contrast, Sentinel and SentinelGlobal performed almost on par with FedAvg.

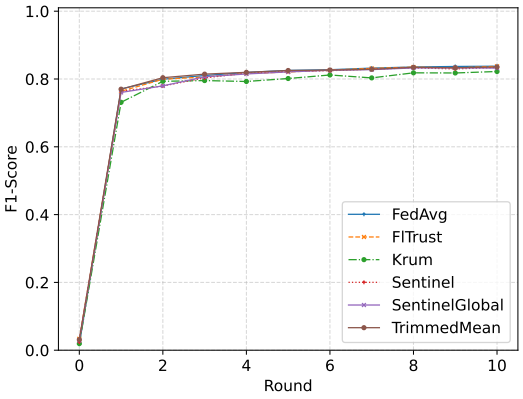
Figure 6.2 visualizes the FL process for the complete process. For MNIST and FMNIST, almost all nodes reached the final F1-score after round 1. For CIFAR10, the increased complexity of the learning task becomes noticeable. The F1-score continuously increased with each round. On the same dataset, Krum appeared to converge faster, but becomes outpaced after round 2 and its performance levels off. With these results presented as the baseline performance, the task difficulty is further referred to as the maximum F1-score a participant can achieve with the given model architecture on a specific dataset, regardless of the aggregation algorithm in place. Therefore, the relative task difficulty for the chosen datasets can be put in the following increasing order: MNIST, FMNIST, CIFAR10. Furthermore, the performance reduction from the baseline is calculated by the difference between the results under any attack and the baseline performance of FedAvg.



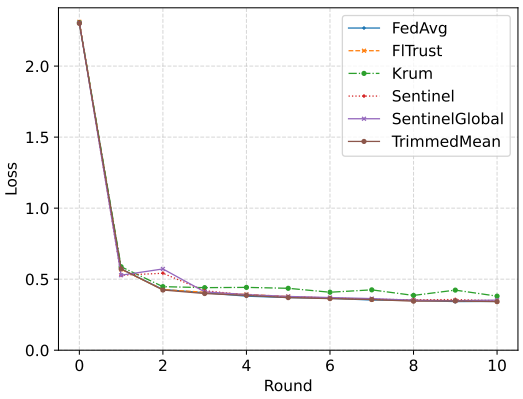
(a) MNIST F1-Score



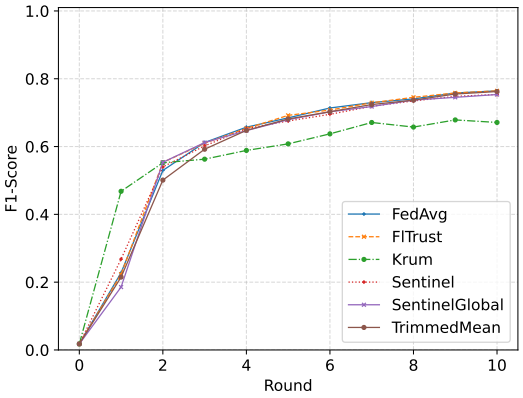
(b) MNIST Loss



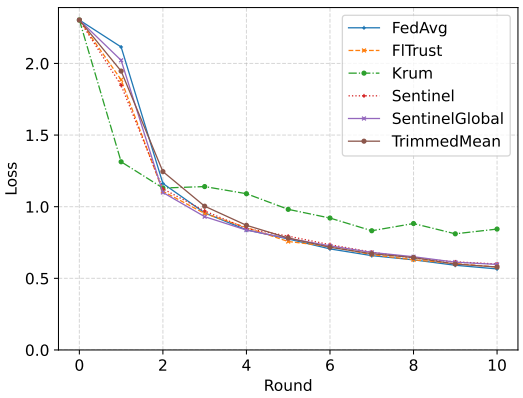
(c) FMNIST F1-Score



(d) FMNIST Loss



(e) CIFAR10 F1-Score



(f) CIFAR10 Loss

Figure 6.2: Baseline performance for MNIST, FMNIST and CIFAR10 for 10 rounds, with the F1-score on the y-axis and the round progression on the x-axis.

Table 6.2: Baseline F1-score performance for MNIST, FMNIST and CIFAR10 after 10 rounds in terms of mean and SEM on the local test dataset.

(a) Mean F1-score after round 10.

F1-Score

	MNIST	FMNIST	CIFAR10
FedAvg	0.953±0.037	0.838±0.027	0.764±0.022
FlTrust	0.952±0.037	0.837±0.022	0.764±0.025
Krum	0.935±0.046	0.822±0.021	0.671±0.029
Sentinel	0.951±0.041	0.834±0.025	0.754±0.018
SentinelGlobal	0.954±0.039	0.832±0.024	0.753±0.021
TrimmedMean	0.952±0.039	0.835±0.025	0.762±0.018

(b) Mean Loss after round 10.

Loss

	MNIST	FMNIST	CIFAR10
FedAvg	0.076±0.064	0.343±0.061	0.566±0.034
FlTrust	0.079±0.058	0.343±0.065	0.578±0.052
Krum	0.152±0.107	0.381±0.082	0.844±0.062
Sentinel	0.079±0.063	0.348±0.064	0.600±0.046
SentinelGlobal	0.080±0.064	0.353±0.066	0.597±0.048
TrimmedMean	0.079±0.061	0.342±0.070	0.579±0.044

6.2.2 Model Poisoning

In the untargeted model poisoning attack, malicious actors train a local model on unmodified training data. But before the model is sent to their neighbors at each round, the weights are randomly altered with salt noise. The NR remained fixed at 80% for all configurations. Instead, the PNR was varied with the ratios 10%, 50% and 80% for all datasets. Table 6.3 summarizes the performance in terms of the F1-score for the three datasets and each PNR configuration after round 10. Sentinel and SentinelGlobal performed the best, with a maximum F1-score reduction of 0.04 from the baseline performance with FedAvg, observed on CIFAR10 and a PNR of 80%. Figure 6.3 illustrates the defense effectiveness of all aggregators for the three datasets. Sentinel and SentinelGlobal were unaffected by the model poisoning attack for all PNR, when compared to the other aggregators. Both FedAvg and TrimmedMean, with an F1-score lower than 0.02, were unable to learn from the dataset under a model poisoning attack with any PNR. For a PNR of 10%, only Sentinel, SentinelGlobal, FlTrust and Krum were able to defend against model poisoning on MNIST and FMNIST. On CIFAR10, FlTrust cannot reach any acceptable F1-score for any PNR. Krum generally performs slightly worse than Sentinel and SentinelGlobal, but could not defend against a model poisoning attack when more than the majority of nodes are malicious, on all three datasets. To conclude, other investigated aggregators can be severely affected by model poisoning, while Sentinel and SentinelGlobal remain robust for any number of attackers involved in the FL process.

Table 6.3: F1-score performance in terms of mean and SEM under targeted model poisoning for MNIST, FMNIST and CIFAR10 after 10 rounds for all PNR configurations.

(a) MNIST F1-Score

F1-Score

PNR	10	50	80
FedAvg	0.018±0.000	0.018±0.000	0.018±0.000
FlTrust	0.837±0.007	0.018±0.000	0.074±0.004
Krum	0.934±0.005	0.919±0.007	0.029±0.001
Sentinel	**0.952±0.004**	0.949±0.007	0.937±0.005
SentinelGlobal	0.949±0.004	**0.957±0.006**	**0.949±0.010**
TrimmedMean	0.018±0.000	0.018±0.000	0.019±0.000

(b) FMNIST F1-Score

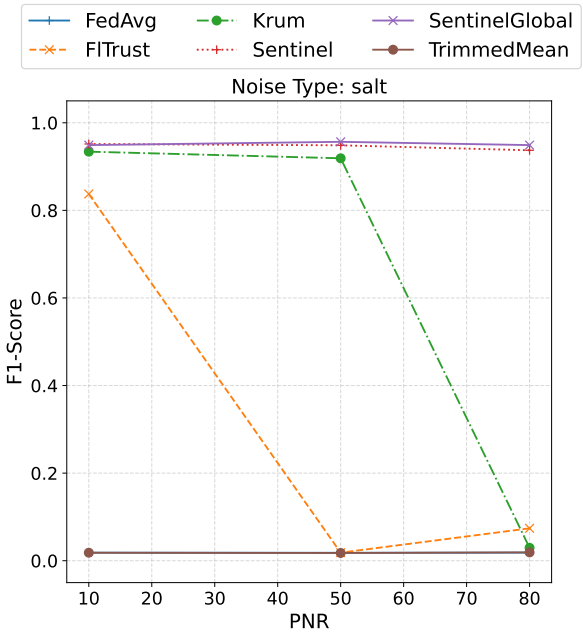
F1-Score

PNR	10	50	80
FedAvg	0.017±0.000	0.018±0.000	0.017±0.000
FlTrust	0.759±0.002	0.018±0.000	0.067±0.000
Krum	0.816±0.003	0.790±0.005	0.028±0.001
Sentinel	0.834±0.003	0.836±0.004	**0.830±0.002**
SentinelGlobal	**0.838±0.003**	**0.838±0.005**	0.814±0.004
TrimmedMean	0.018±0.000	0.018±0.000	0.018±0.000

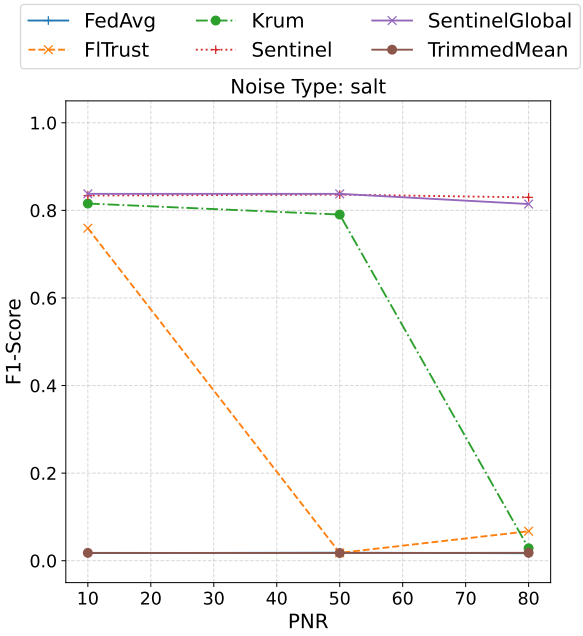
(c) CIFAR10 F1-Score

F1-Score

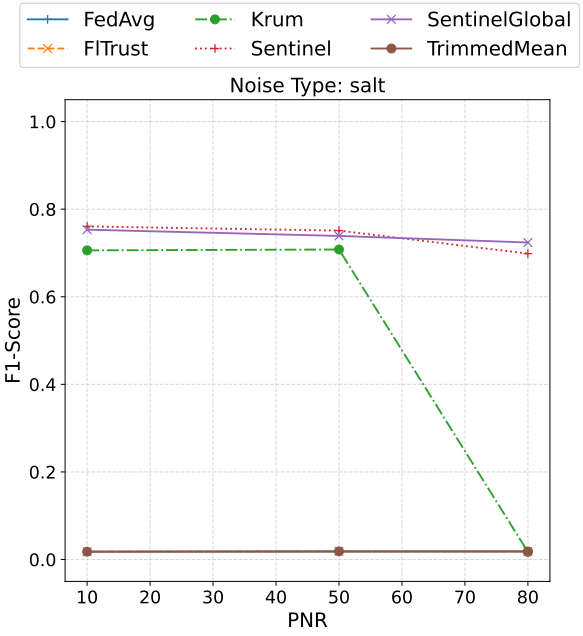
PNR	10	50	80
FedAvg	0.018±0.000	0.018±0.000	0.019±0.000
FlTrust	0.018±0.000	0.019±0.000	0.018±0.001
Krum	0.706±0.003	0.708±0.003	0.018±0.001
Sentinel	**0.761±0.002**	**0.751±0.004**	0.699±0.007
SentinelGlobal	0.753±0.003	0.739±0.002	**0.724±0.002**
TrimmedMean	0.018±0.000	0.019±0.000	0.018±0.001



(a) MNIST



(b) FMNIST



(c) CIFAR10

Figure 6.3: Performance under model poisoning for MNIST, FMNIST, CIFAR10 after 10 rounds, with the F1-score on the y-axis and an increasing PNR on the x-axis.

6.2.3 Untargeted Label Flipping

With this attack, the adversaries modify the local training data by randomly altering the labels of their data samples. Thereby, malicious actors learn a different task than benign ones. The aim is to degrade the performance of the model from benign actors, *i.e.*, provoking random mispredictions. As declared in Table 6.1, the PNR was varied with a value of 10%, 50% and 80%, combined with selected PSR values of 30%, 50% and 100%. Table 6.4 and 6.4 summarize the result of the effectiveness of untargeted label flipping. With the lowest PSR of 30%, most aggregators were not affected by the label flipping of adversaries, for any PNR. Only the performance of FlTrust severely declined with a high number of attackers (PNR = 80%): in case of the FMNIST dataset, the F1-score dropped by over 50% compared to the baseline performance. Similar results can be observed for a PSR of 50%. Surprisingly, FlTrust was not affected in the CIFAR10 experiments. Instead, benign nodes applying the Krum aggregation performed the lowest for all PSR configurations, followed by FedAvg. With the highest PSR of 100%, the effectiveness of untargeted label flipping becomes apparent: while all aggregators remained robust for a low PNR of 10%, none of the selected reference algorithms were able to defend the attack for a PNR of 80%. Only Sentinel and SentinelGlobal could maintain their baseline performance for any attack configuration. The maximum performance drop for the aggregators proposed in this work was observed for Sentinel on CIFAR10 and the configuration PSR = 100%, PNR = 80%, with a F1-score reduction of approximately 8.4%.

Table 6.4: F1-score performance in terms of mean and SEM under untargeted label flipping for MNIST, FMNIST and CIFAR10 after 10 rounds for all PNR & PSR configurations.

(a) MNIST

F1-Score

PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	**0.955±0.005**	0.950±0.005	0.954±0.004	0.948±0.007	0.947±0.009	0.860±0.009	0.923±0.012	0.903±0.004	0.038±0.002
FlTrust	0.906±0.006	0.913±0.007	0.911±0.007	0.870±0.006	0.833±0.005	0.516±0.003	0.605±0.002	0.441±0.004	0.054±0.002
Krum	0.934±0.005	0.935±0.005	0.927±0.006	0.950±0.008	0.937±0.007	0.907±0.009	**0.958±0.005**	0.839±0.004	0.001±0.000
Sentinel	0.952±0.005	0.956±0.004	0.949±0.005	0.962±0.003	0.955±0.007	**0.955±0.005**	0.922±0.006	0.925±0.008	0.927±0.006
SentinelGlobal	0.951±0.005	**0.957±0.004**	**0.958±0.004**	0.945±0.007	**0.959±0.006**	0.946±0.007	0.944±0.004	**0.930±0.006**	**0.954±0.010**
TrimmedMean	0.950±0.004	0.952±0.005	0.948±0.004	**0.970±0.003**	0.957±0.004	0.772±0.005	0.947±0.014	0.922±0.016	0.025±0.007

(b) FMNIST

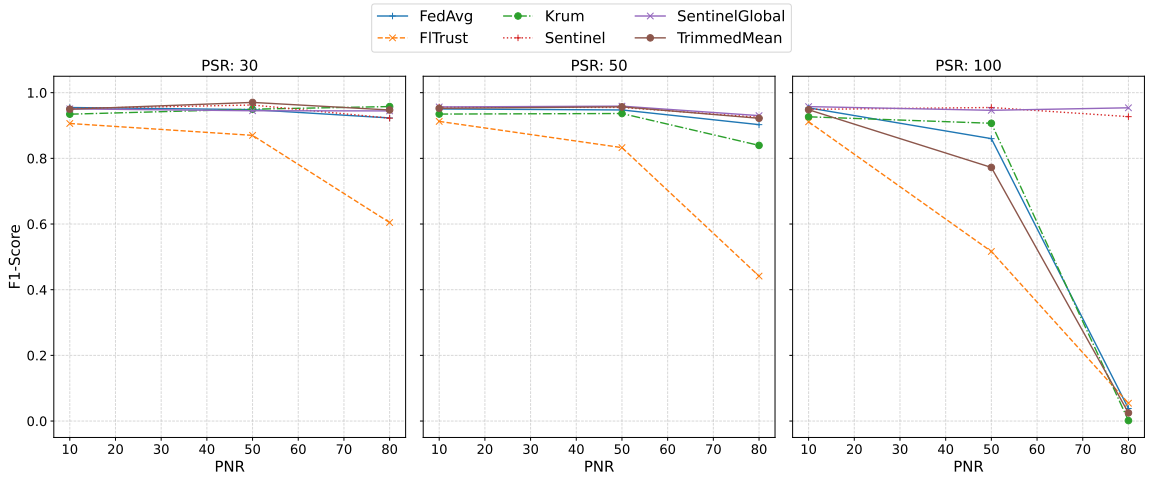
F1-Score

PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	0.831±0.003	0.829±0.003	0.829±0.003	**0.828±0.005**	0.818±0.003	0.629±0.018	0.809±0.006	0.804±0.000	0.016±0.002
FlTrust	0.774±0.002	0.777±0.003	0.779±0.003	0.603±0.003	0.496±0.001	0.330±0.004	0.406±0.007	0.297±0.006	0.041±0.002
Krum	0.807±0.003	0.810±0.003	0.820±0.003	0.785±0.004	0.792±0.003	0.007±0.001	0.804±0.002	0.803±0.005	0.009±0.001
Sentinel	0.833±0.003	0.833±0.003	**0.834±0.004**	0.822±0.003	**0.838±0.003**	**0.833±0.004**	0.833±0.003	**0.818±0.000**	0.826±0.002
SentinelGlobal	**0.837±0.003**	**0.838±0.003**	0.834±0.003	0.816±0.004	0.837±0.003	0.827±0.004	0.824±0.002	0.817±0.003	**0.840±0.001**
TrimmedMean	0.829±0.003	0.831±0.002	0.833±0.003	0.821±0.004	0.806±0.005	0.404±0.020	**0.840±0.001**	0.808±0.009	0.009±0.003

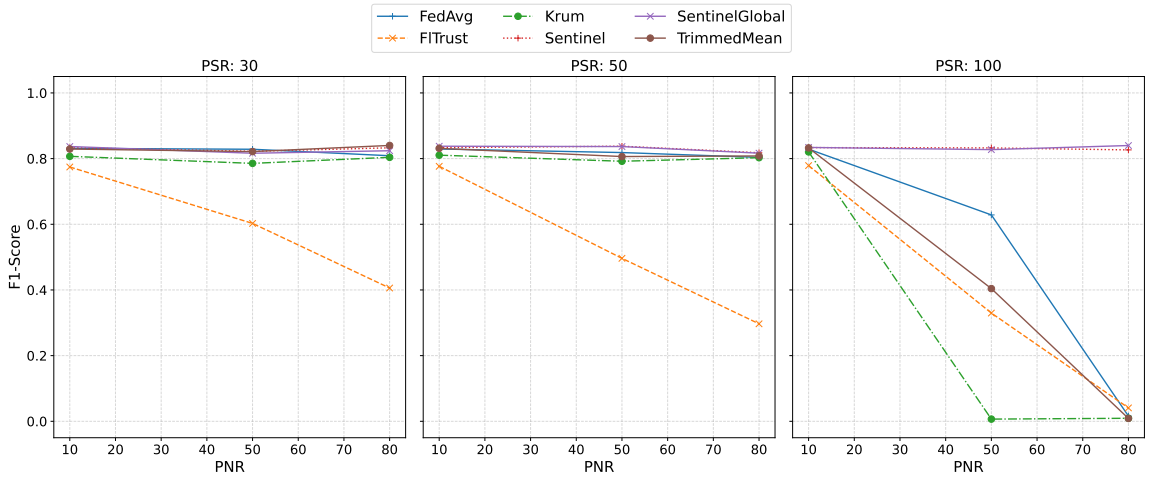
(c) CIFAR10

F1-Score

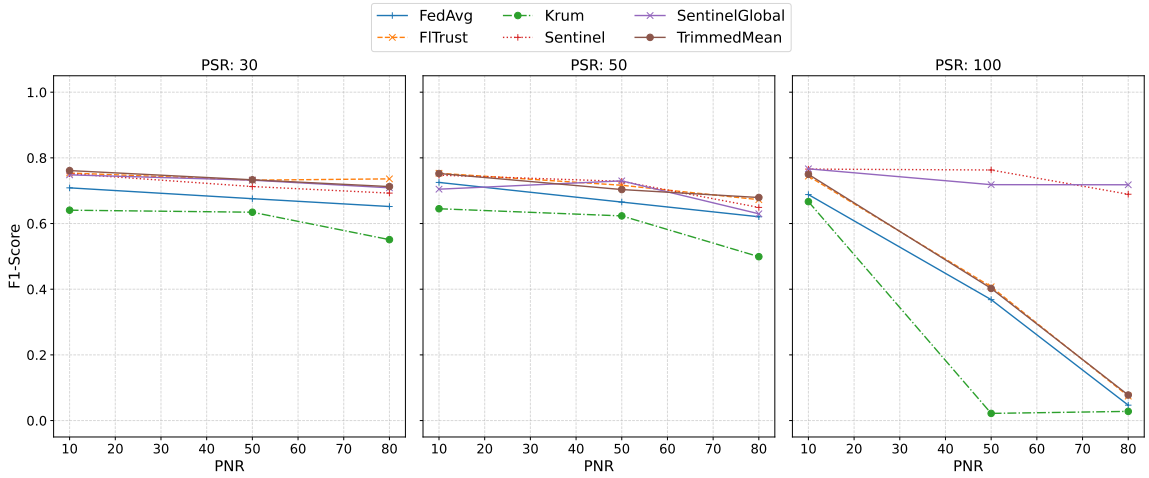
PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	0.709±0.002	0.725±0.003	0.688±0.002	0.676±0.003	0.665±0.004	0.368±0.003	0.652±0.009	0.621±0.012	0.047±0.005
FlTrust	0.754±0.003	**0.753±0.003**	0.744±0.002	0.732±0.004	0.717±0.004	0.407±0.004	**0.736±0.013**	0.673±0.005	0.075±0.004
Krum	0.641±0.004	0.645±0.003	0.667±0.002	0.634±0.002	0.623±0.004	0.022±0.001	0.551±0.001	0.499±0.006	0.028±0.002
Sentinel	0.752±0.003	0.747±0.002	0.766±0.002	0.713±0.002	0.728±0.005	**0.763±0.006**	0.693±0.003	0.649±0.005	0.689±0.005
SentinelGlobal	0.748±0.002	0.705±0.001	**0.767±0.002**	0.732±0.003	**0.730±0.005**	0.718±0.004	0.709±0.004	0.630±0.013	**0.718±0.000**
TrimmedMean	**0.761±0.003**	0.752±0.002	0.750±0.003	**0.733±0.003**	0.704±0.001	0.402±0.002	0.713±0.002	**0.679±0.005**	0.078±0.000



(a) MNIST



(b) FMNIST



(c) CIFAR10

Figure 6.4: Performance under untargeted label flipping for MNIST, FMNIST, CIFAR10 and different PSR configurations after 10 rounds, with the F1-score on the y-axis and increasing PNR on the x-axis.

6.2.4 Targeted Label Flipping

The targeted objective of label flipping does not aim at degrading the model performance of benign participants, Instead, the attack aims at provoking a misclassification of a selected source label l_{scr} to a target label l_t. The success of this poisoning attack is measured by the amount of test data samples labeled with the source label classified as the target label by the final models of benign participants. This ratio is represented by the ASR_{LF}, as defined in Section 4.1. The results of the experiments with this attack are reported in Table 6.5 and Figure 6.5. At first glance, none of the investigated aggregators were affected by targeted label flipping with the lowest PNR of 10%. Hence, a single malicious actor was not able to effectively attack the benign nodes of the FL scenario. Additionally, the attack effectiveness appears to decrease with increasing task difficulty: on MNIST, the highest ASR_{LF} achieved is 0.969 for Krum. In contrast, the malicious participants only reached a maximum ASR_{LF} of 0.486 when attacking benign nodes applying Krum.

From the given results, it can also be observed that the PSR does not have a crucial impact on the ASR_{LF} for FlTrust and TrimmedMean. Furthermore, benign participants performing an aggregation with Krum were only able to defend a maximum amount of 50% of malicious actors on MNIST. On FMNIST and CIFAR10, Krum performed remarkably worse than the other aggregation algorithm. On the other hand, Sentinel and SentinelGlobal effectively defended against the label flipping attack: for all PNR and PSR configurations on MNIST, the ASR_{LF} remained below 0.02. For FMNIST, the attackers were not able to introduce any misclassification to the target label, *i.e.*, the ASR_{LF} remained 0 for all experiments.

Only in experiments on the CIFAR10 dataset, a small attack effectiveness was observed with a maximum ASR_{LF} of 0.078 for Sentinel with a configuration of PNR= 80% and PSR= 100%. Furthermore, SentinelGlobal appeared to perform better than Sentinel on CIFAR10, with an ASR_{LF} reduction of 80% and 100% for a PSR= 50% and a PSR=100%, respectively. This can be explained by the exclusion of untrusted nodes after round r_α by SentinelGlobal and thereby preventing an aggregation of malicious, but highly similar, poisoned models.

Figure 6.6 visualizes these results in the form of the confusion matrix computed on the local test dataset of FMNIST for each node representing the participant n_0. For the reference algorithms in the Figures 6.6a to 6.6d, it is observable that almost all data samples with source label $l_{src=3}$ “Dress” were incorrectly predicted as the target label $l_{t=7}$ “Shirt”. Some samples were classified with a label other than the target label or the source label, but none of them were correctly classified. In contrast, a success of the targeted label flipping attack was not observed for Sentinel and SentinelGlobal, where none of the test samples having label $l_{scr=3}$ were classified as the target label $l_{t=7}$. The confusion matrices for MNIST and CIFAR10 illustrate the same idea and are thus not reported in this section.

Table 6.5: ASR_{LF} performance in terms of mean and SEM under targeted label flipping for MNIST, FMNIST and CIFAR10 after 10 rounds for all PNR & PSR configurations.

(a) MNIST

ASR-LF

PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	**0.004±0.001**	0.008±0.001	0.008±0.002	0.152±0.009	0.729±0.023	0.207±0.044	0.981±0.005	0.962±0.001	0.935±0.009
FlTrust	0.012±0.002	0.017±0.003	0.021±0.004	0.391±0.011	0.544±0.020	0.412±0.020	0.843±0.005	0.904±0.018	0.976±0.002
Krum	0.007±0.001	0.008±0.001	0.010±0.002	0.952±0.007	0.014±0.002	0.004±0.002	0.944±0.005	0.965±0.002	0.955±0.001
Sentinel	0.006±0.001	**0.004±0.001**	**0.006±0.001**	0.008±0.002	**0.002±0.001**	**0.004±0.001**	**0.005±0.002**	**0.005±0.002**	**0.010±0.000**
SentinelGlobal	0.008±0.002	0.005±0.001	**0.006±0.001**	**0.004±0.001**	0.004±0.002	0.006±0.001	0.010±0.005	**0.005±0.002**	0.015±0.002
TrimmedMean	0.010±0.001	0.009±0.002	0.009±0.002	0.428±0.043	0.469±0.038	0.629±0.047	0.948±0.016	0.965±0.001	0.954±0.009

(b) FMNIST

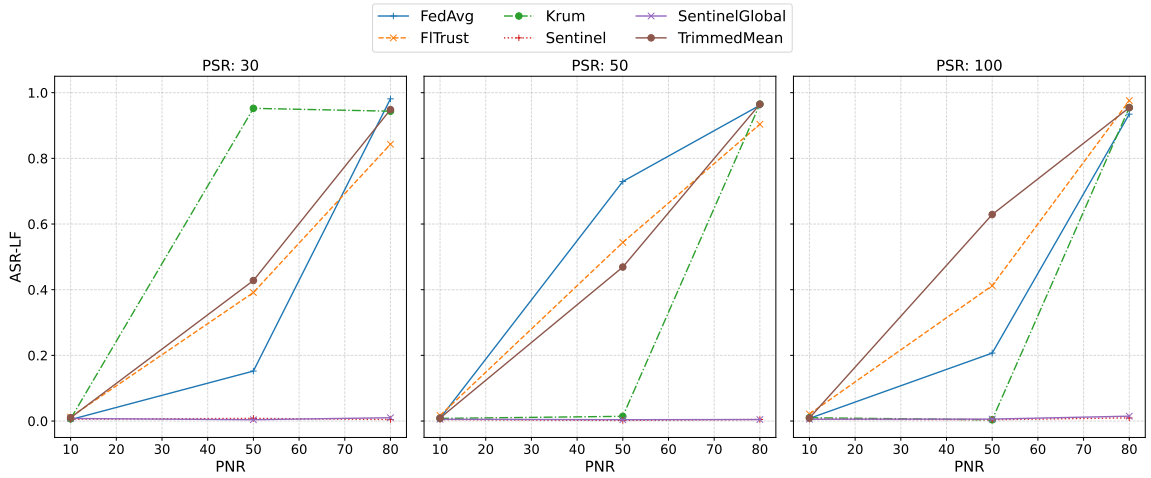
ASR-LF

PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	0.329±0.028	0.162±0.024	0.109±0.020	0.802±0.008	0.767±0.001	0.752±0.003
FlTrust	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	0.398±0.005	0.385±0.011	0.314±0.010	0.769±0.013	0.760±0.005	0.783±0.005
Krum	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	0.868±0.006	0.907±0.005	0.847±0.006	0.868±0.013	0.795±0.008
Sentinel	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**
SentinelGlobal	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**
TrimmedMean	**0.000±0.000**	**0.000±0.000**	**0.000±0.000**	0.255±0.033	0.156±0.013	0.139±0.025	0.729±0.004	0.831±0.001	0.704±0.002

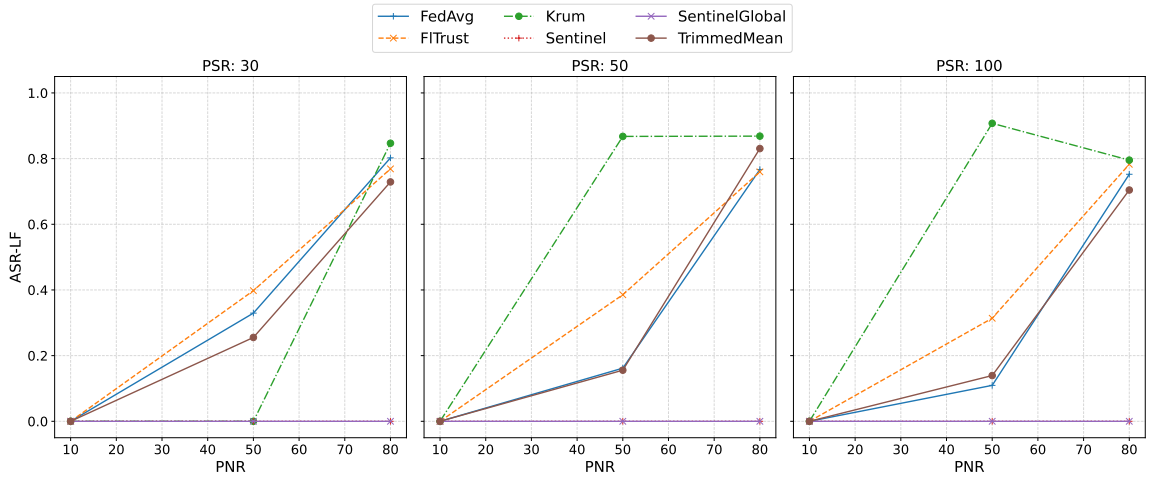
(c) CIFAR10

ASR-LF

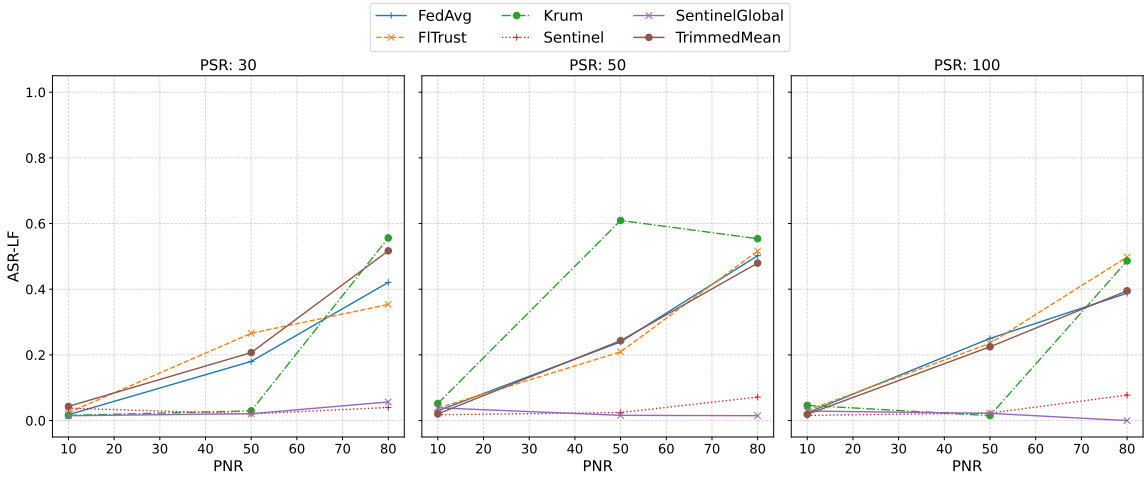
PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	0.018±0.002	0.030±0.003	0.024±0.002	0.180±0.013	0.240±0.010	0.250±0.019	0.421±0.012	0.502±0.006	0.388±0.029
FlTrust	0.024±0.003	0.038±0.004	0.031±0.002	0.266±0.006	0.209±0.007	0.235±0.010	0.353±0.036	0.516±0.014	0.498±0.008
Krum	0.016±0.002	0.052±0.003	0.046±0.004	0.030±0.002	0.609±0.010	**0.015±0.004**	0.556±0.019	0.554±0.010	0.486±0.014
Sentinel	0.038±0.002	**0.018±0.001**	**0.016±0.001**	**0.020±0.003**	0.024±0.004	0.023±0.002	**0.040±0.015**	0.072±0.004	0.078±0.006
SentinelGlobal	**0.015±0.001**	0.039±0.002	0.029±0.003	0.021±0.004	**0.016±0.004**	0.022±0.002	0.057±0.004	**0.015±0.007**	**0.000±0.000**
TrimmedMean	0.043±0.003	0.020±0.002	0.019±0.002	0.207±0.003	0.243±0.011	0.225±0.010	0.517±0.019	0.480±0.037	0.395±0.015



(a) MNIST

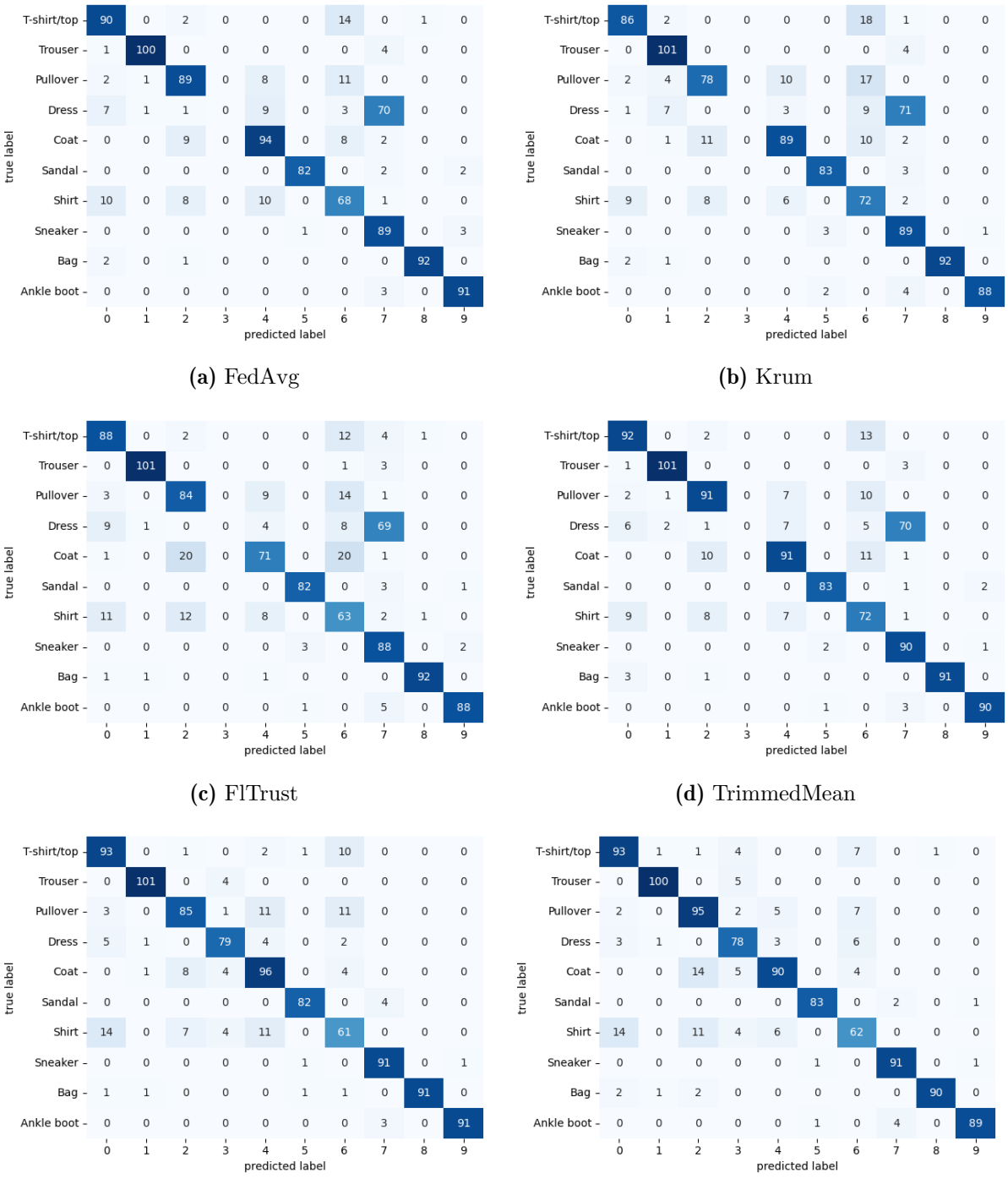


(b) FMNIST



(c) CIFAR10

Figure 6.5: Performance under targeted label flipping for MNIST, FMNIST, CIFAR10 and different PSR configurations after 10 rounds, with the ASR_{LF} on the y-axis and increasing PNR on the x-axis.



(a) FedAvg **(b)** Krum

(c) FlTrust **(d)** TrimmedMean

(e) Sentinel **(f)** SentinelGlobal

Figure 6.6: The confusion matrix at round 10 of participant n_0 for each aggregator, computed on the test dataset under targeted label flipping on FMNIST. The values represent the absolute number of samples with their true and predicted label. PNR = 80%, PSR = 100%.

6.2.5 Backdoor

For the last attack in this section, the effectiveness of targeted sample poisoning, *i.e.*, the backdoor attack defined in 4.1, will be discussed. In this attack, malicious participants try to provoke a misclassification to a predefined target label l_t by using an artificial trigger. In this work, this trigger is represented by the form of an "X" on any image of the three selected datasets. Table 6.6 and Figure 6.7 put the results of the experiments with different aggregators and attack configurations into perspective. To measure the success of the adversarial participants in each FL scenario, the BA is computed on a copy of the test dataset, where each sample is marked with the selected trigger. The higher the BA, the more successful the attack.

A single attacker could not successfully incorporate the backdoor into the model of benign participants for all experiments. With a PNR of 10% and a PSR of 30%, the highest BA of 0.124 is reported for FlTrust on MNIST. However, all algorithms performed almost equally well in low adversarial environments. In regard to MNIST & FMNIST and a sufficient number of attackers (PNR $>$ 10%), it becomes apparent that the lowest PSR of 30% already fully poisoned the models of benign participants, when any of the reference aggregators was used. In contrast, Sentinel and SentinelGlobal achieved the lowest BA overall for any PNR $>$ 10%. Without considering CIFAR10, the highest BA achieved on the aggregators proposed in this work was observed for Sentinel on FMNIST and the configuration PSR = 50%, PNR = 80%, with a value of 0.151. SentinelGlobal indicated an improved performance over Sentinel for all experiments with PSR $\geq$ 30%, similar to the observations in targeted label flipping in Section 6.2.4.

Figure 6.8 illustrates the performance difference between the proposed aggregation mechanisms and the reference aggregators: Figures 6.8a to 6.8d visualize the backdoor confusion matrix for each experiment on FMNIST with a fixed configuration of PSR = 100%, PNR = 80%. In this case, the backdoor confusion matrix is computed on the triggered test dataset using the model of benign participant n_0. The confusion matrices for MNIST with the aforementioned configuration are very similar to the ones of FMNIST and are thus not reported.

With the CIFAR10 dataset, however, the results look different: firstly, it is notable that in general the attacker success is lower with the increased task complexity of CIFAR10. Overall, the average BA is higher in MNIST and FMNIST for any PNR $>$ 30%. The BA increased linearly with the increase of the PNR, whereas the increase in PSR seemed to have little impact. In terms of the aggregation algorithm in place, the results appeared to be highly variable, such that the overall best performing algorithm cannot be determined. This is especially valid for Krum, which reported the lowest BA for a PSR $>$ 30% and a PNR = 50%, but the weakest defense results for higher PNR. Furthermore, Figure 6.9 also highlights that Sentinel and SentinelGlobal could not reliably defend against the backdoor attack on CIFAR10. For all aggregators, almost all backdoor test samples are classified as the target label $l_{t=3}$. In conclusion, none of the investigated aggregation algorithms could effectively defend against poisoning attacks on CIFAR10.

Table 6.6: BA in terms of mean and SEM under targeted sample poisoning (backdoor attack) for MNIST, FMNIST and CIFAR10 after 10 rounds for all PNR & PSR configurations.

(a) MNIST

BA

PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	0.067±0.007	0.103±0.010	0.087±0.008	0.960±0.003	0.953±0.004	0.950±0.005	0.973±0.003	0.992±0.001	0.974±0.006
FlTrust	0.124±0.012	0.088±0.010	0.097±0.010	0.932±0.007	0.948±0.004	0.944±0.006	0.994±0.000	0.966±0.000	0.986±0.001
Krum	0.028±0.003	0.011±0.001	0.005±0.001	0.852±0.010	0.875±0.010	0.897±0.006	0.960±0.011	0.933±0.016	0.972±0.008
Sentinel	**0.004±0.001**	**0.005±0.001**	0.011±0.001	0.037±0.003	**0.005±0.001**	0.028±0.007	**0.005±0.001**	0.007±0.002	**0.003±0.000**
SentinelGlobal	0.005±0.000	0.006±0.001	**0.003±0.000**	**0.004±0.001**	0.008±0.001	**0.007±0.001**	0.012±0.002	**0.007±0.000**	**0.003±0.000**
TrimmedMean	0.068±0.007	0.111±0.008	0.074±0.007	0.948±0.005	0.960±0.005	0.924±0.005	0.961±0.010	0.983±0.004	0.984±0.002

(b) FMNIST

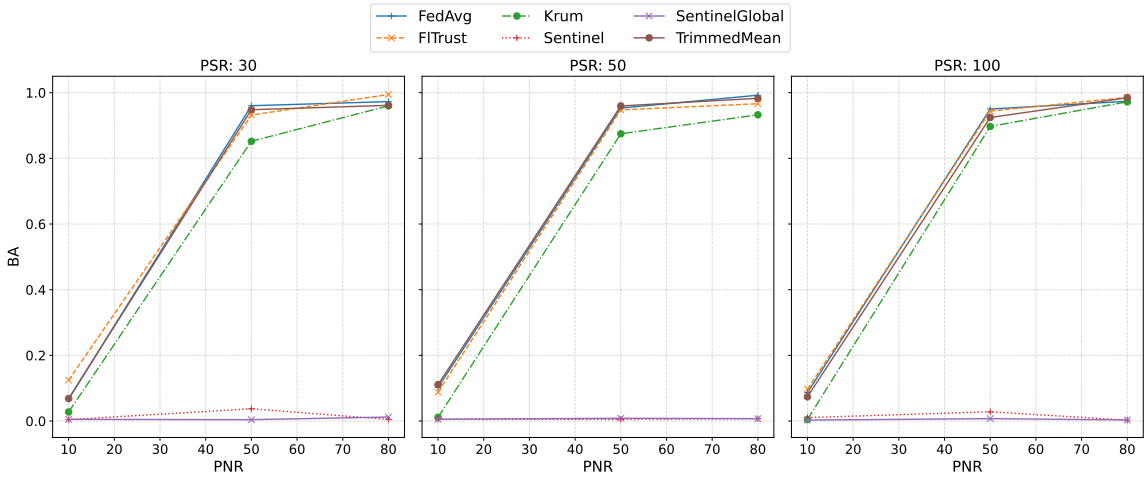
BA

PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	0.049±0.001	0.059±0.001	0.098±0.002	0.649±0.003	0.767±0.004	0.703±0.002	0.789±0.005	0.775±0.004	0.766±0.004
FlTrust	0.057±0.001	0.083±0.002	0.065±0.002	0.671±0.003	0.632±0.003	0.686±0.004	0.791±0.002	0.779±0.005	0.740±0.002
Krum	**0.009±0.001**	**0.011±0.001**	0.012±0.001	0.676±0.003	0.676±0.003	0.848±0.002	0.728±0.006	0.564±0.002	0.781±0.008
Sentinel	0.013±0.001	0.017±0.001	0.013±0.001	**0.021±0.001**	0.035±0.001	0.024±0.001	**0.021±0.002**	0.151±0.000	0.118±0.001
SentinelGlobal	0.012±0.001	0.012±0.001	**0.009±0.001**	**0.021±0.001**	**0.022±0.001**	**0.018±0.001**	0.057±0.000	**0.089±0.001**	**0.017±0.000**
TrimmedMean	0.052±0.001	0.065±0.002	0.078±0.002	0.760±0.005	0.646±0.003	0.680±0.003	0.798±0.005	0.758±0.003	0.730±0.002

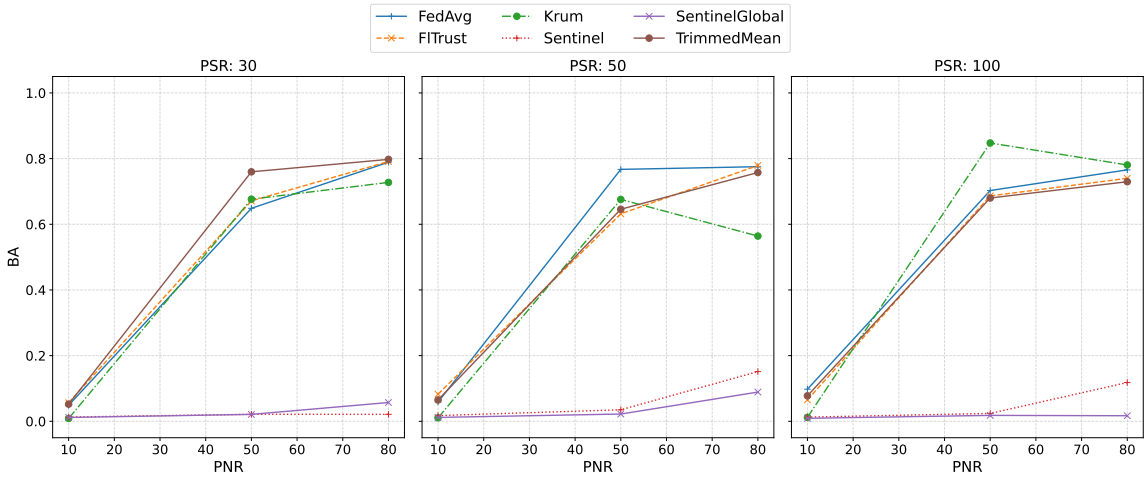
(c) CIFAR10

BA

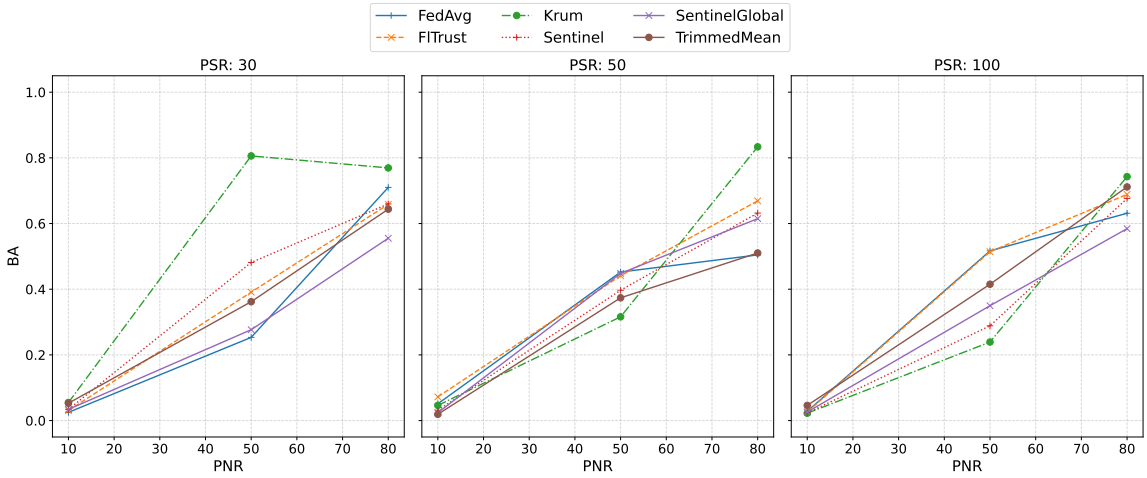
PNR	**10**			**50**			**80**		
PSR	**30**	**50**	**100**	**30**	**50**	**100**	**30**	**50**	**100**
FedAvg	**0.025±0.001**	0.051±0.001	0.028±0.001	**0.253±0.005**	0.452±0.002	0.517±0.002	0.710±0.010	**0.504±0.003**	0.631±0.001
FlTrust	0.031±0.001	0.072±0.001	0.026±0.001	0.391±0.005	0.441±0.005	0.514±0.003	0.658±0.006	0.669±0.006	0.689±0.017
Krum	0.055±0.001	0.046±0.001	**0.022±0.001**	0.806±0.004	**0.316±0.004**	**0.239±0.001**	0.769±0.007	0.834±0.008	0.743±0.004
Sentinel	0.034±0.001	0.031±0.001	0.023±0.001	0.481±0.006	0.397±0.006	0.288±0.006	0.660±0.021	0.632±0.012	0.676±0.014
SentinelGlobal	0.034±0.001	0.024±0.001	0.026±0.001	0.277±0.009	0.448±0.007	0.350±0.005	**0.555±0.001**	0.615±0.001	**0.584±0.013**
TrimmedMean	0.053±0.002	**0.019±0.000**	0.046±0.001	0.362±0.004	0.374±0.005	0.415±0.005	0.644±0.007	0.510±0.008	0.712±0.001



(a) MNIST

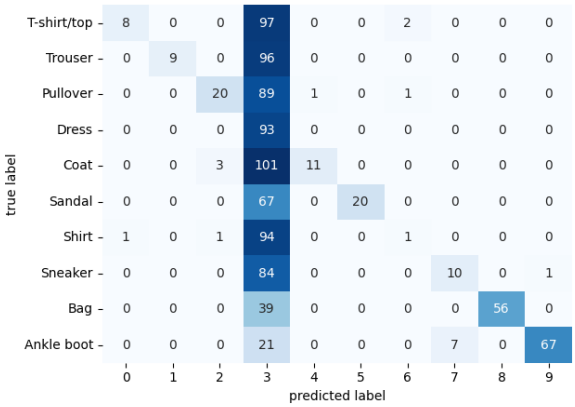


(b) FMNIST

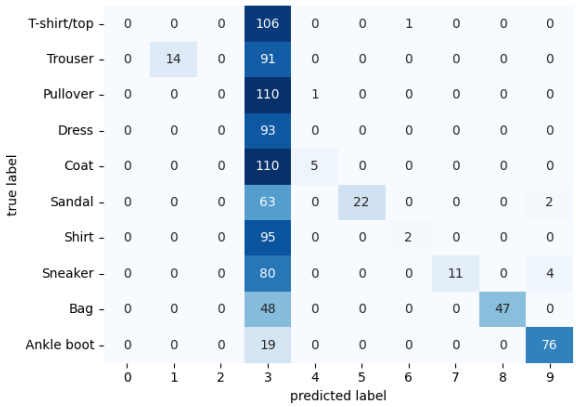


(c) CIFAR10

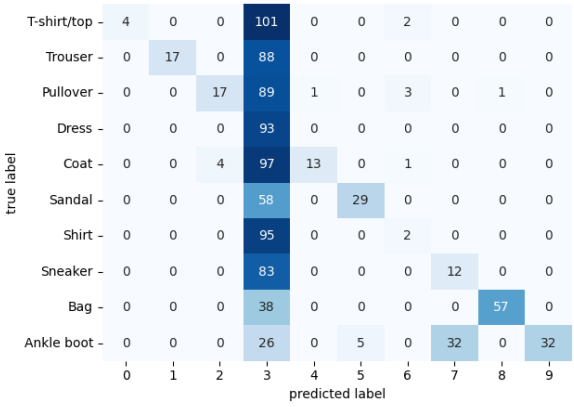
Figure 6.7: Performance targeted samples poisoning (backdoor attack) for MNIST, FMNIST, CIFAR10 and different PSR configurations after 10 rounds, with the BA on the y-axis and increasing PNR on the x-axis.



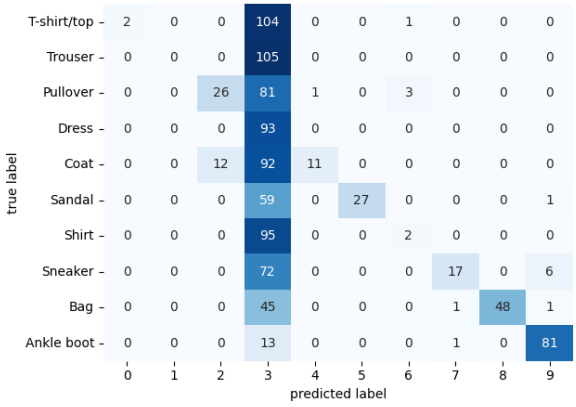
(a) FedAvg



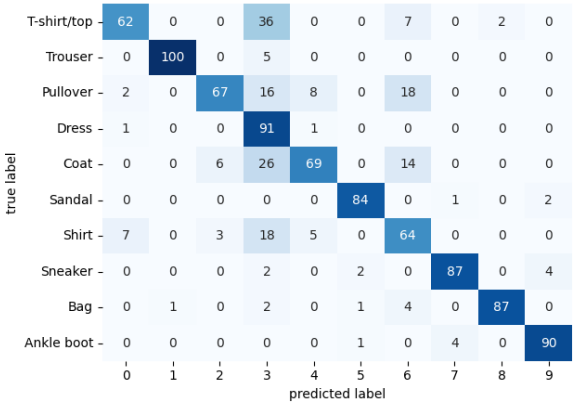
(b) Krum



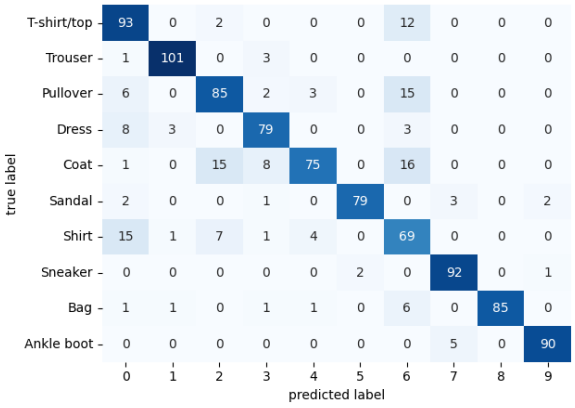
(c) FlTrust



(d) TrimmedMean

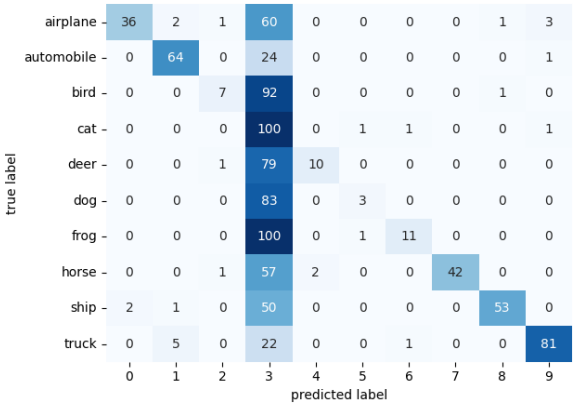


(e) Sentinel

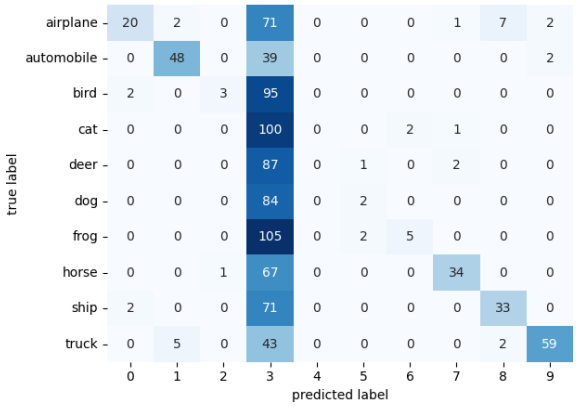


(f) SentinelGlobal

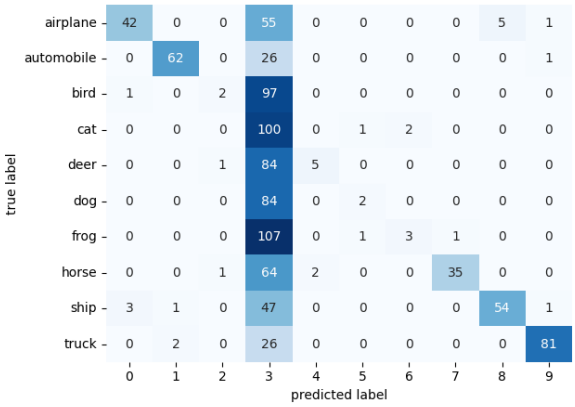
Figure 6.8: The backdoor confusion matrix after round 10 of participant n_0 for each aggregator, computed on the triggered test dataset under backdoor attack on FMNIST. The values represent the absolute number of samples with their true and predicted label. PNR = 80%, PSR = 100%.



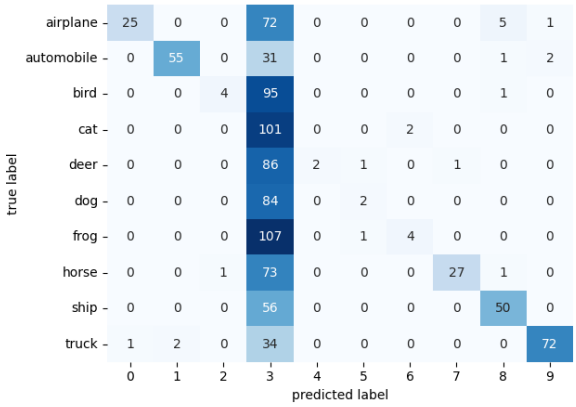
(a) FedAvg



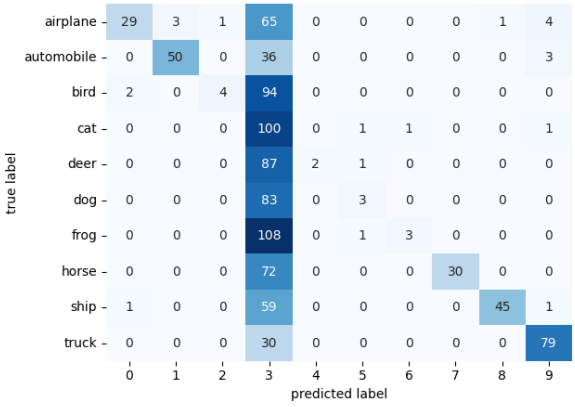
(b) Krum



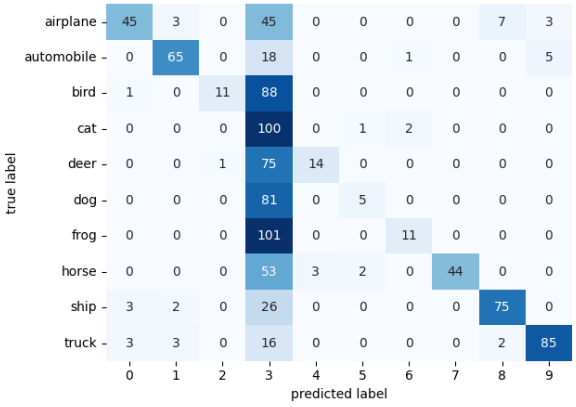
(c) FlTrust



(d) TrimmedMean



(e) Sentinel



(f) SentinelGlobal

Figure 6.9: The backdoor confusion matrix after round 10 of participant n_0 for each aggregator, computed on the triggered test dataset under backdoor attack on CIFAR10. The values represent the absolute number of samples with their true and predicted label. PNR = 80%, PSR = 100%.

6.2.6 Compute Resources

A resource-efficient aggregation mechanism is crucial to the scalability of DFL. In the following, the compute resources used by each FL scenario with a corresponding aggregator in use are analyzed. Therefore, the resources consumed by a single node were collected. The raw data is available on the GitHub repository[5]. For each experiment, the benign participant n_0 is used to compare the metrics between aggregators, datasets and attack types. It is important to note that the resource consumption generally depends on the configuration of the ML framework, such as the number of workers used for data loading[6]. For the experiments in this section, this configuration remained the same. The following metrics are chosen to report the performance of each aggregation algorithm:

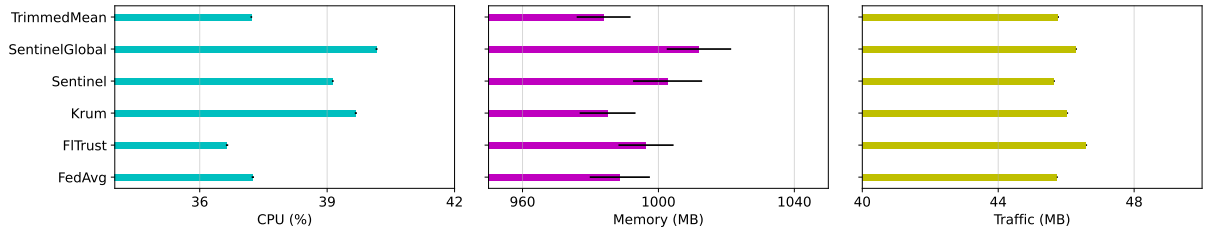
- CPU (%): is reported as the mean utilization in percent of all CPUs available to a node (*i.e.*, 48 in case of the CPU environment listed in 6.1.4), averaged over 10 rounds. Additionally, the SEM is reported.
- GPU (%): is reported as the utilization in percent, averaged over all rounds. As a node in Fedstellar only accesses one GPU, the metrics are computed for a single processor. The GPU instances are only used by CIFAR10. The mean GPU utilization is reported together with the SEM.
- Memory (MB): is reported as the average amount of RAM consumed over all rounds, measured in MB. If the node used a GPU, the VRAM is reported instead.
- Traffic (MB): refers to the total amount of messages sent to other nodes summed up to the completion of round 10, measured in MB. No corresponding error is reported, as this metric is an absolute total.

As a reference, 6.10 illustrates the baseline performance in terms of computing resources in the benign environment, *i.e.*, in the absence of any malicious participants. Numerical values are available in 6.7. The results on resource usage are reported for the FMNIST and CIFAR10 dataset to be able to compare the results with GPU and CPU metrics.

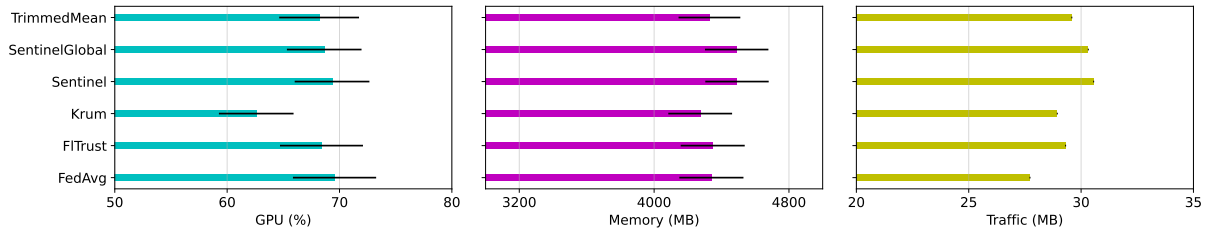
In a benign environment, all aggregators utilized a similar amount of compute resources: The processor utilization was determined within a range of approximately 37% to 41% for CPU and 62% to 70% for GPU. The highest processor utilization is reached by SentinelGlobal, with 40.164 % and 68.639 % for CPU and GPU, respectively. In terms of memory, SentinelGlobal again reported the highest consumption with 1011 MB on FMNIST, whereas the highest average on CIFAR10 was reached by Sentinel with 4491 MB. However, the difference to the best performing aggregator is negligibly small, as highlighted in Figure 6.10. The amount of total traffic reached a range of approx. 45 – 47 MB for FMNIST and 27 – 31 MB for CIFAR10. The reduced amount of traffic in CIFAR10 can be explained by the size of the models exchanged. In contrast, CIFAR10 required more processor utilization and memory.

[5] evaluation/metrics/resources

[6] https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader



(a) Baseline FMNIST



(b) Baseline CIFAR10

Figure 6.10: The baseline compute metrics in a benign environment for FMNIST and CIFAR10. For CPU and memory, the mean and SEM over all rounds is shown. Traffic represents the total amount of messages sent.

Table 6.7: The baseline compute metrics in a benign environment for FMNIST and CIFAR10. For CPU and memory, the mean and SEM over all rounds is shown. Traffic represents the total amount of messages sent.

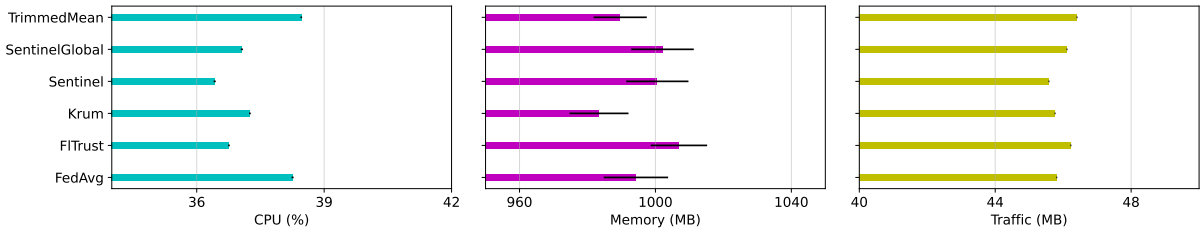
(a) Baseline FMNIST

	CPU (%)	Memory (MB)	Traffic (MB)
FedAvg	37.251±0.025	988.638±8.220	45.724
FlTrust	36.644±0.027	996.347±8.849	46.586
Krum	39.676±0.019	985.039±7.933	46.027
Sentinel	39.133±0.022	1002.696±9.485	45.637
SentinelGlobal	40.164±0.021	1011.923±10.164	46.285
TrimmedMean	37.215±0.027	983.873±8.113	45.758

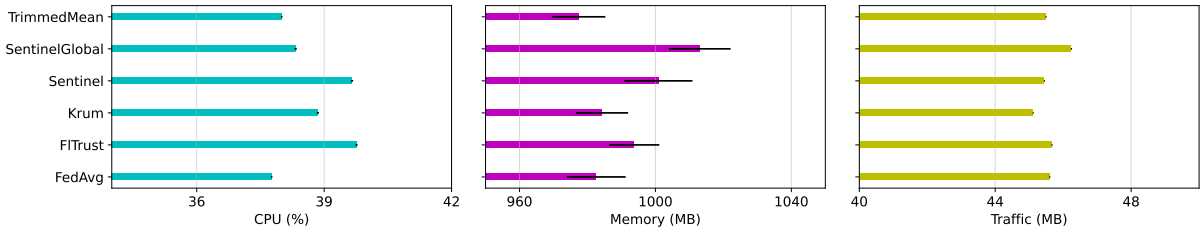
(b) Baseline CIFAR10

	GPU (%)	Memory (MB)	Traffic (MB)
FedAvg	69.559±3.320	4339.889±188.203	27.703
FlTrust	68.396±3.322	4347.667±189.277	29.289
Krum	62.589±3.552	4273.368±183.244	28.919
Sentinel	69.337±3.693	4491.737±189.929	30.537
SentinelGlobal	68.639±3.702	4489.895±190.232	30.299
TrimmedMean	68.185±3.324	4328.222±188.080	29.566

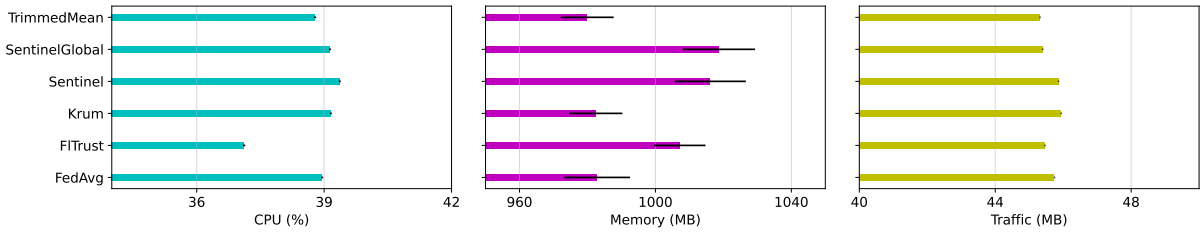
Figure 6.11 and Table 6.8 summarize the results for the resource consumption of each attack and dataset. Similar to the baseline, there were no notable differences in resource usage. For all attacks, the CPU usage stretched between 36% and 41%, whereas the GPU utilization was between 50% and 70%. In terms of memory, the investigated aggregation algorithms used between 979 and 1018 MB of RAM on FMNIST and between 3819 and 4486 MB of VRAM on CIFAR10. Whereas three is no observed pattern for CPU utilization, TrimmedMean, Krum and FedAvg generally use less RAM than FlTrust, Sentinel and SentinelGlobal for all attacks on CIFAR10. Overall, SentinelGlobal reported a slight performance improvement over Sentinel, with a maximum reduction of processor GPU of approx. 14% (absolute) and a reduction of RAM consumption of approx. 500 MB (absolute) on CIFAR10 for targeted label flipping. The maximum reduction for CPU utilization was below 2%. However, the differences in resource usage were generally very small between all aggregators, for both RAM/VRAM and CPU/GPU. Additionally, Sentinel and SentinelGlobal both showed a slightly increased amount of traffic. This is expected, due to the additional metrics and trust scores that are exchanged between nodes.



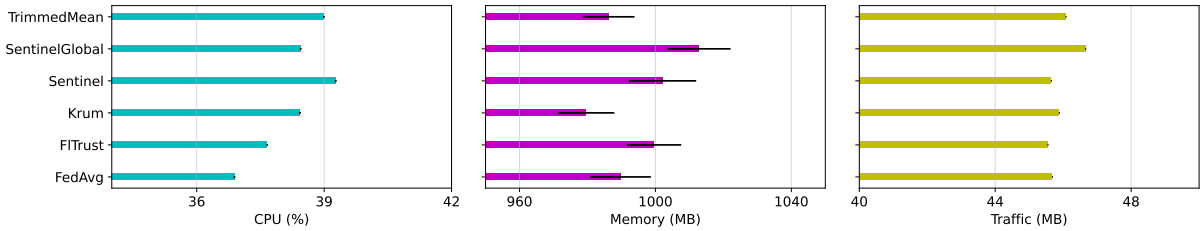
(a) Model Poisoning



(b) Untargeted Label Flipping

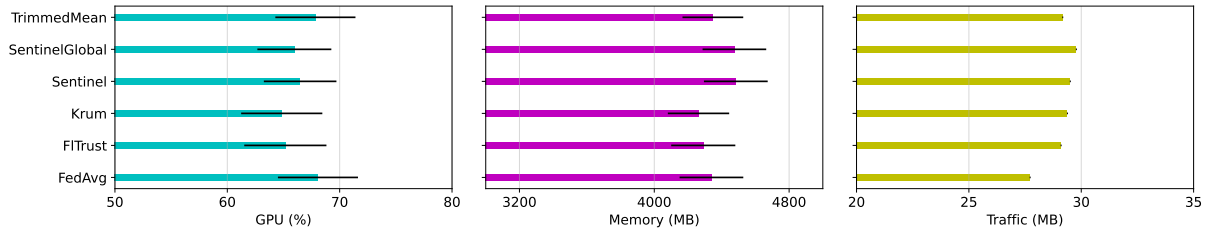


(c) Targeted Label Flipping

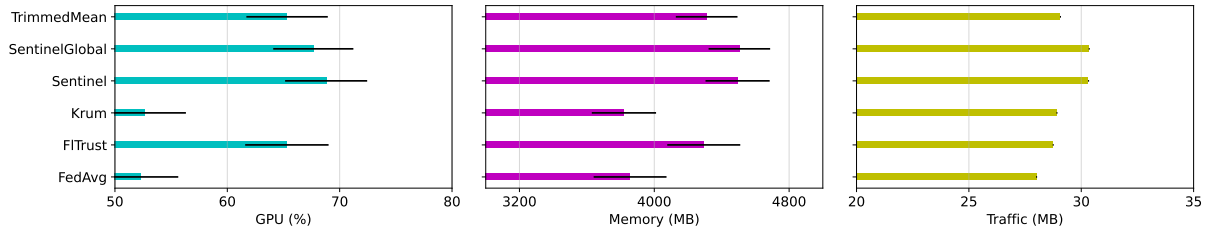


(d) Backdoor

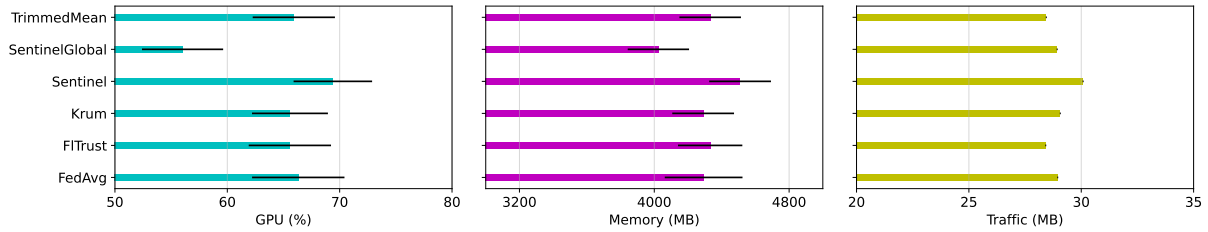
Figure 6.11: The compute metrics for all attacks with PNR = 80%, PSR = 100% (except model poisoning) on FMNIST. For CPU and memory, the mean and SEM over all rounds is shown. Traffic represents the total amount of messages sent.



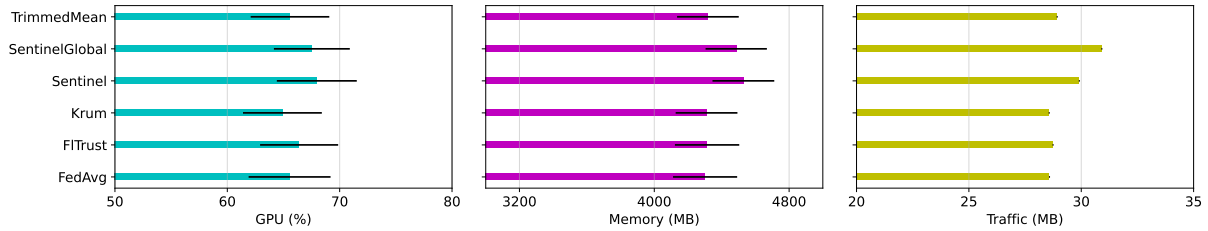
(a) Model Poisoning



(b) Untargeted Label Flipping



(c) Targeted Label Flipping



(d) Backdoor

Figure 6.12: The compute metrics for all attacks with PNR = 80%, PSR = 100% (except model poisoning) on CIFAR10. For CPU and memory, the mean and SEM over all rounds is shown. Traffic represents the total amount of messages sent.

Table 6.8: The compute metrics for all attacks with PNR = 80%, PSR = 100% (except model poisoning) on CIFAR10 and FMNIST. For CPU and memory, the mean and SEM over all rounds is shown. Traffic represents the total amount of messages sent.

(a) Model Poisoning, FMNIST

	CPU (%)	Memory (MB)	Traffic (MB)
FedAvg	38.257±0.020	994.257±8.680	45.797
FlTrust	36.760±0.024	1006.921±9.479	46.213
Krum	37.251±0.019	983.391±7.833	45.747
Sentinel	36.423±0.025	1000.556±9.208	45.568
SentinelGlobal	37.063±0.024	1002.074±9.158	46.097
TrimmedMean	38.462±0.019	989.628±8.299	46.393

(b) Model Poisoning, CIFAR10

	GPU (%)	Memory (MB)	Traffic (MB)
FedAvg	68.068±3.290	4339.222±188.807	27.703
FlTrust	65.165±3.612	4290.632±181.961	29.082
Krum	64.845±3.565	4262.421±180.241	29.363
Sentinel	66.482±3.657	4483.789±190.263	29.490
SentinelGlobal	65.968±3.562	4475.368±188.983	29.760
TrimmedMean	67.841±3.230	4347.611±189.144	29.155

(c) Untargeted Label Flipping, FMNIST

	CPU (%)	Memory (MB)	Traffic (MB)
FedAvg	37.765±0.024	982.587±7.680	45.590
FlTrust	39.764±0.016	993.797±8.657	45.658
Krum	38.852±0.016	984.295±7.879	45.103
Sentinel	39.656±0.024	1000.868±9.098	45.432
SentinelGlobal	38.330±0.027	1013.030±10.020	46.230
TrimmedMean	38.003±0.022	977.379±7.349	45.481

(d) Untargeted Label Flipping, CIFAR10

	GPU (%)	Memory (MB)	Traffic (MB)
FedAvg	52.281±3.355	3856.429±216.172	27.991
FlTrust	65.302±3.567	4293.526±182.627	28.730
Krum	52.624±3.712	3819.929±216.632	28.897
Sentinel	68.796±3.652	4494.368±190.317	30.304
SentinelGlobal	67.653±3.692	4504.842±190.673	30.333
TrimmedMean	65.321±3.623	4310.789±183.003	29.053

(e) Targeted Label Flipping, FMNIST

	CPU (%)	Memory (MB)	Traffic (MB)
FedAvg	38.956±0.021	982.875±7.805	45.735
FlTrust	37.118±0.021	1007.164±9.658	45.450
Krum	39.160±0.021	982.465±7.763	45.927
Sentinel	39.369±0.023	1016.137±10.631	45.855
SentinelGlobal	39.136±0.021	1018.687±10.469	45.387
TrimmedMean	38.788±0.027	979.949±7.569	45.304

(f) Targeted Label Flipping, CIFAR10

	GPU (%)	Memory (MB)	Traffic (MB)
FedAvg	66.316±3.611	4292.842±181.976	28.931
FlTrust	65.568±3.377	4331.632±183.226	28.399
Krum	65.577±3.659	4289.947±182.524	29.045
Sentinel	69.393±3.662	4509.158±190.844	30.062
SentinelGlobal	56.019±4.111	4024.357±230.544	28.906
TrimmedMean	65.919±3.494	4331.737±183.229	28.423

(g) Backdoor, FMNIST

	CPU (%)	Memory (MB)	Traffic (MB)
FedAvg	36.890±0.026	989.737±8.290	45.670
FlTrust	37.661±0.021	999.613±8.921	45.540
Krum	38.430±0.020	979.613±7.497	45.876
Sentinel	39.274±0.021	1002.111±9.222	45.641
SentinelGlobal	38.450±0.021	1012.892±9.887	46.650
TrimmedMean	38.993±0.018	986.351±7.986	46.072

(h) Backdoor, CIFAR10

	GPU (%)	Memory (MB)	Traffic (MB)
FedAvg	65.543±3.378	4301.526±181.895	28.566
FlTrust	66.393±3.508	4313.316±183.287	28.723
Krum	64.905±3.499	4310.105±182.953	28.551
Sentinel	67.966±3.474	4528.684±190.859	29.891
SentinelGlobal	67.525±3.646	4486.211±190.202	30.895
TrimmedMean	65.579±3.557	4318.263±183.306	28.907

6.3 Discussion

From the results in Section 6.2, it becomes clear that poisoning attacks are a serious threat to DFL. Most aggregation algorithms in the chosen subset of this work cannot effectively defend against malicious participants, especially with an increasing PNR. With more than the majority of the participants being benign, only Sentinel and its extended version SentinelGlobal could reliably defend against any number of attackers.

However, it is important to consider the focus of this work. The DFL process can be seen from two perspectives: the global view is the perspective of an administrator, that is responsible for the setup of the FL process. The goal in the view of the global perspective is to evaluate the FL scenario as a whole, such that all participants conclude with a performant and benign model. Additionally, a global observer should be able to distinguish between benign and malicious participants. In the case where the majority of participants are malicious, this distinction becomes unreasonable, since a global observer considers the task the majority aims to learn as benign. With a PNR $> 50\%$, the attackers' goal will become the global goal, as the majority trains a poisoned model. In contrast to the global perspective, the focus of this thesis is the local node perspective. This view considers the local objective individually, which is to obtain a performant, benign model as a local node. Thus, from a local perspective, a node needs to be able to mitigate poisoning attacks from any number of neighbors. Its ultimate goal is to conclude the FL process with a trusted model that generalizes well.

The increased threat of a large attacker group was observed for untargeted model poisoning attacks for all investigated datasets. In terms of the untargeted label flipping attack, a high PSR is required to effectively impact the FL process. For the targeted version of label flipping, however, a low amount of altered labels is already sufficient to provoke a misclassification in the models of benign participants. Sentinel and SentinelGlobal were not successfully impacted by label flipping.

Nevertheless, the results in Section 6.2.5 highlight the threat of backdoor introduced by targeted sample poisoning: it appears to be fairly effortless in terms of PNR and PSR for a group of attackers to incorporate the backdoor in the model of the benign participants. While Sentinel and its variant were able to effectively defend the backdoor attack on MNIST and FMNIST, none of the investigated algorithms indicated any defensive capabilities on CIFAR10. Thus, an increased task complexity appears to correlate with the effectiveness of the defense mechanisms to some extent, at least for backdoor attacks. This highlights the need for an investigation of further, more complex datasets. In the following, the effectiveness of Sentinel and SentinelGlobal are investigated in detail for each attack. Furthermore, the choice of the algorithm parameters of Sentinel will be analyzed. Lastly, the computational advantages of SentinelGlobal over Sentinel is highlighted.

6.3.1 Metric Effectiveness

To investigate the effectiveness of the proposed aggregators in detail, a subset of defensive metrics is analyzed. Therefore, all four attacks in Section 6.2 were evaluated with a PNR

= 50%, combined with a PSR $\in \{30, 50, 100\}$. Since the attacker behavior in terms of metrics is similar for other PNR, they are not investigated in this section. For each attack, the metrics are reported by the benign participant n_0 on all neighbor models. All data can be retrieved from the GitHub repository[7] of Festellar-Sync. It is distinguished between models classified as benign or malicious, *i.e.*, the corresponding parameters are received from an actual honest or malicious neighbor. Further, the following metrics are analyzed for each set of model parameters P_j received by a neighboring node n_j: cosine similarity S_j, average bootstrap loss $\bar{l}_j$, and the aggregation weight w_j (before normalization).

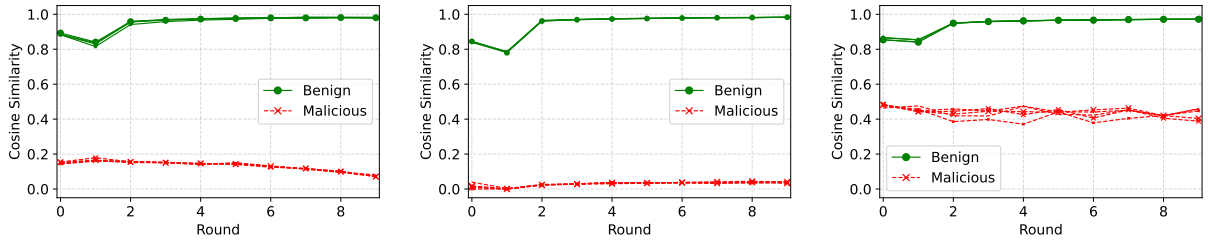
Model Poisoning

In case the FL scenario is attacked by model poisoning, the similarity filtering of Sentinel was sufficient to exclude the adversarial models from aggregation, for all three datasets. Figure 6.13 illustrates the difference in cosine similarity between benign and malicious models. The benign models can clearly be separated from the malicious models: after round 2, the models received from benign neighbors almost reach a cosine similarity of 1. This was observed for all datasets. For malicious models, the cosine similarity was constantly below 0.2 for MNIST and FMNIST, and below 0.5 for CIFAR10. Hence, with a similarity threshold $\tau_S = 0.5$, all malicious models were filtered by Sentinel. Moreover, it is interesting to see that the cosine similarity was always positive, except for the first two rounds on FMNIST. This also explains the low performance of FlTrust under model poisoning in Section 6.2.2: since its algorithm aggregates any model with a positive cosine similarity proportionally, the malicious models can take up a substantial part of the final model, especially when the poisoned models exhibit a large cosine similarity despite being poisoned. The small drop in similarity at round 1 is due to the aggregation not being active at round 0, as this round is a diffusion round in Fedstellar.

Since the cosine similarity of malicious models is always below the predefined threshold, their bootstrap validation loss is never computed. Consequently, as all malicious models had received a cosine similarity lower than the threshold, they were never validated on the bootstrap dataset. This also led to an aggregation weight $w_i = 0$ for all malicious models. In contrast, the benign models take up relatively equal shares of the aggregated model. In these experiments, the validation loss is not computed for any filtered models. But other experiments have shown that the bootstrap validation loss rises exponentially for models altered with random weights generation. This means in case a malicious model exhibits a cosine similarity higher than the threshold, it would still be assigned an aggregation weight of $w_i = 0$ due to the bootstrap loss mapping.

In this thesis, only the salt noise type was investigated with a fixed PNR = 80%. Lower PNR did not appear to be effective against most aggregation algorithms. It may be interesting to experiment with other noise types and investigate the behavior of Sentinel and SentinelGlobal. For example, a Gaussian noise exhibits a higher variability in changed weights, which may provoke different effects in an FL scenario.

[7] evaluation/metrics/sentinel



(a) Cosine similarity MNIST

(b) Cosine similarity FMNIST

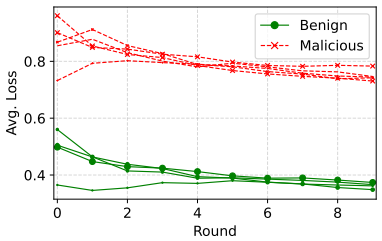
(c) Cosine similarity CIFAR10

Figure 6.13: The cosine similarity of models received from benign and malicious neighbors at participant n_0 under model poisoning.

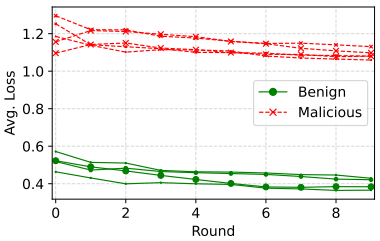
Untargeted Label Flipping

For the untargeted variant of label flipping, the cosine similarity was not sufficient to filter malicious models. Figure 6.14 illustrates the average bootstrap validation loss values and Figure 6.15 the resulting aggregation weights for FMNIST and CIFAR10. Only for FMNIST and a PSR = 100%, the average loss of malicious models remained constant after the first round, which means they were not further evaluated. For all other scenarios, there is a clear difference in terms of loss between benign and malicious models. However, this difference appeared to diminish with a decreasing PSR, since fewer labels were flipped. In CIFAR10 and PSR set to 30% and 50%, it can be observed that the aggregation weights of benign and malicious models become almost equal. Although the poisoned models were aggregated at all rounds, the untargeted label flipping attack vanished for a PSR $< 80\%$, as reported in Section 6.2.3. For the highest PSR, Sentinel was able to successfully distinguish between malicious and benign at any round based on the average bootstrap loss. Nevertheless, these observations illustrate how similar malicious models can be, despite being poisoned.

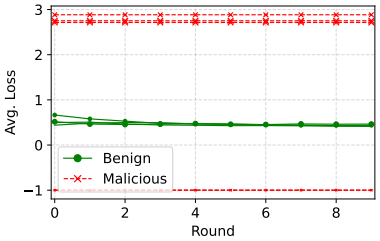
Arguably, the restrictiveness of the *MapLossDistance* function could be increased, *e.g.*, adjusting the damping factor κ, but this would essentially also lead to a higher possibility of excluding benign models that are slightly less similar than the local models. In conclusion, it is difficult to detect poisoned models under label flipping with a low PSR. But due to the low effectiveness of the attack, this deficit can be neglected. Moreover, an attacker cannot anticipate the required PSR to maximize his impact while remaining stealthy and is thus encouraged to apply a maximum PSR. Further investigations on more complex scenarios than CIFAR10 with CNN could be of interest.



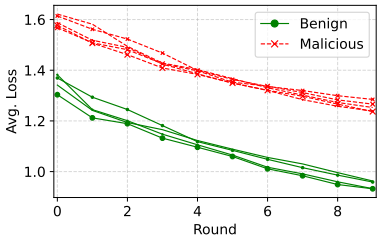
(a) Avg. Loss, PSR: 30% FMNIST



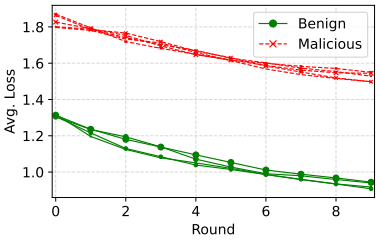
(b) Avg. Loss, PSR: 50% FMNIST



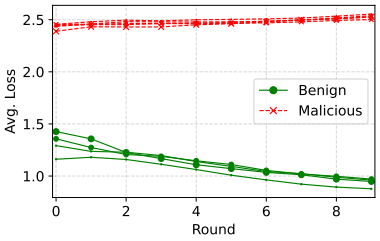
(c) Avg. Loss, PSR: 100% FMNIST



(d) Avg. Loss, PSR: 30% CIFAR10

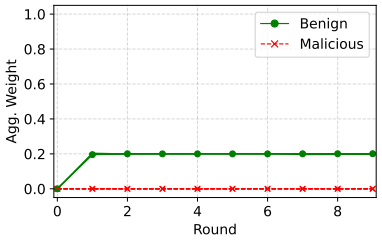


(e) Avg. Loss, PSR: 50% CIFAR10

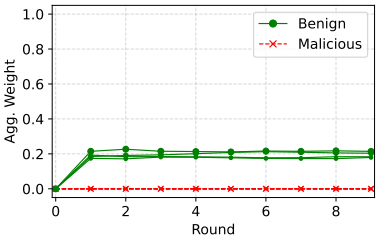


(f) Avg. Loss, PSR: 100% CIFAR10

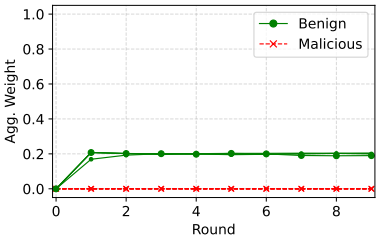
Figure 6.14: The average bootstrap loss of models received from benign and malicious neighbors at participant n_0 under untargeted label flipping on FMNIST and CIFAR10.



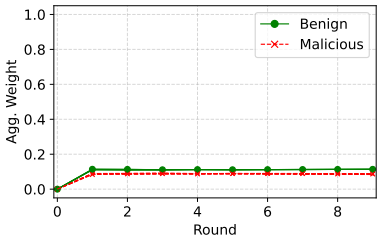
(a) Agg. Weight, PSR: 30% FMNIST



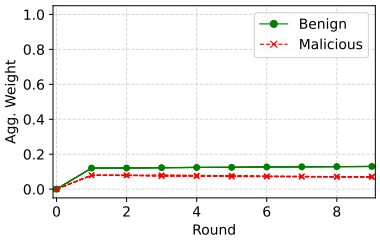
(b) Agg. Weight, PSR: 50% FMNIST



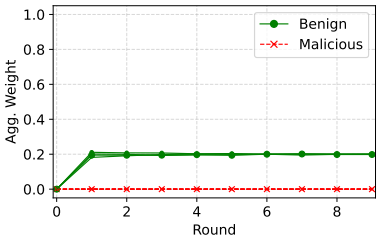
(c) Agg. Weight, PSR: 100% FMNIST



(d) Agg. Weight, PSR: 30% CIFAR10



(e) Agg. Weight, PSR: 50% CIFAR10

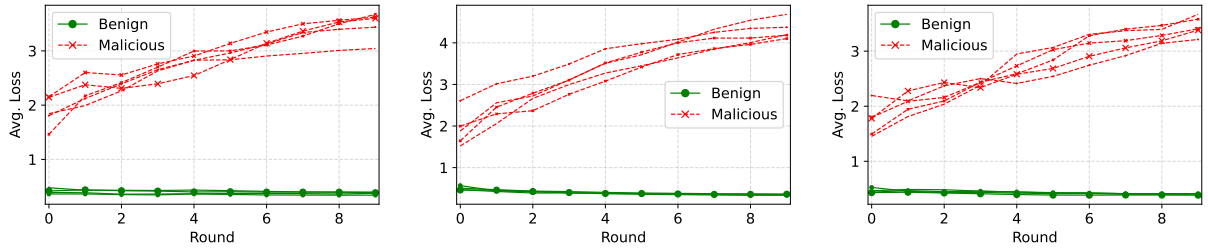


(f) Avg. Weight, PSR: 100% CIFAR10

Figure 6.15: The aggregation weights of models received from benign and malicious neighbors at participant n_0 under untargeted label flipping on FMNIST and CIFAR10.

Targeted Label Flipping

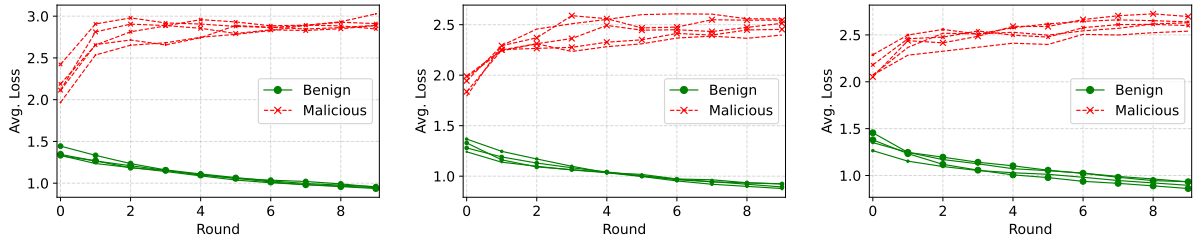
If the label flipping attack is executed with a targeted objective, Sentinel was able to clearly distinguish between malicious and benign models using the bootstrap validation loss. Figure 6.16 displays the average bootstrap loss from the perspective of participant n_0 on all neighbor models for FMNIST and CIFAR10. The metrics on MNIST are very similar, and thus not reported in this section. It can be observed that the average loss of benign models on the bootstrap dataset continuously decreased as the FL scenario progressed. In contrast, the loss of malicious models constantly increased until round 10. Further, there appeared to be no connection between PSR and the loss ratio to benign models. Consequently, all malicious models received an aggregation weight $w_i = 0$, for any dataset and PSR. As a result, the loss distance mapping approach of Sentinel was effective against targeted label flipping in all experiments.



(a) Avg. Loss, PSR: 30%, FMNIST

(b) Avg. Loss, PSR: 50% FMNIST

(c) Avg. Loss, PSR: 100% FMNIST



(d) Avg. Loss, PSR: 30% CIFAR10

(e) Avg. Loss, PSR: 50% CIFAR10

(f) Avg. Loss, PSR: 100% CIFAR10

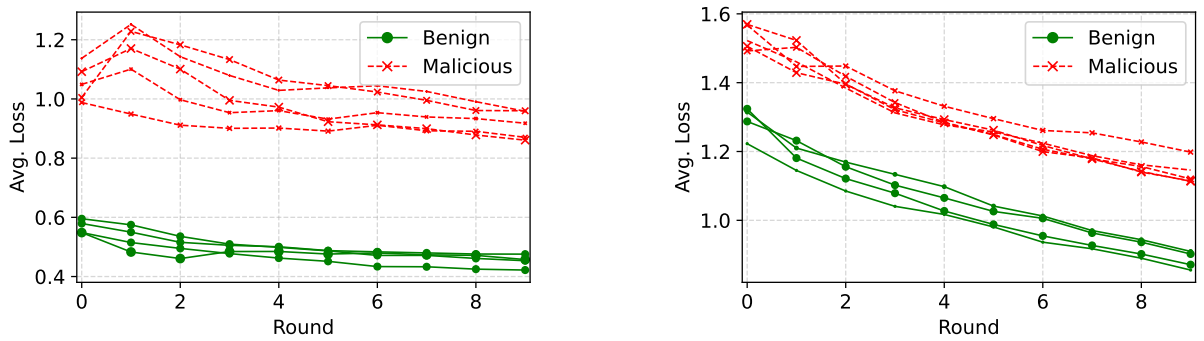
Figure 6.16: The average loss of models received from benign and malicious neighbors at participant n_0 under targeted label flipping on FMNIST and CIFAR10.

Backdoor Attack

In Section 6.2.5, it was pointed out that Sentinel and SentinelGlobal were only able to defend against the backdoor attack on MNIST and FMNIST, for any attack configuration. On CIFAR10, all aggregators fully incorporated the backdoor, whereas the BA appeared to be linearly dependent on the PSR. For a PNR $> 10\%$, even a low PSR of 30% poisoned the models of benign participants with a BA $> 40\%$. Hence, no defense effectiveness was observed for any aggregation algorithm. Similarity measures such as Euclidean distance or cosine similarity are not able to reliably defend against backdoor attacks.

Figure 6.17 and Figure 6.18 display the average bootstrap loss and the aggregation weight on FMNIST and CIFAR10, respectively, with the PSR fixed to 30%. It can be observed that the loss of malicious models is almost twice as high as for benign models. For CIFAR10, on the other hand, this factor is much smaller. Since the adversaries also aggregate benign models during the FL process, their loss decreases at a similar rate as those of honest neighbors. Although Sentinel was able to separate the malicious models based on their loss in FMNIST, this was not possible for CIFAR10, where all received models were assigned similar weights. Hence, participant n_0 was successfully attacked with a BA of 0.27.

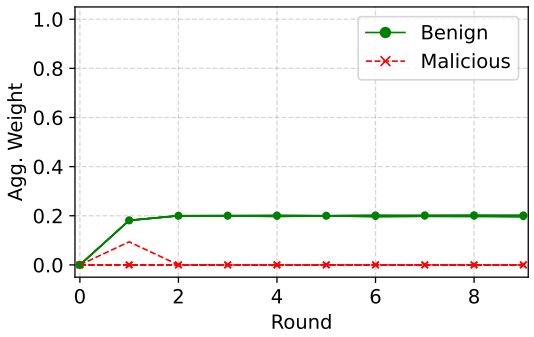
In conclusion, the average bootstrap loss, or a validation loss in general, is not a sufficient metric to detect targeted sample poisoning. One could argue clustering the loss values may be a promising approach. However, only consistent, fully connected networks were analyzed in this work. For other dynamic DFL network configurations, a node may be connected to any number of neighbors between 1 to $|N|$. This makes clustering infeasible, as it cannot be assumed that there will always one cluster of benign and malicious nodes. Alternatively, there may be other metrics that better serve at detecting backdoors. For example, this work only focused on layer-wise average cosine similarity as model similarity measure. It might be possible to identify the weight change responsible for the triggered backdoor classification. Other extensions of Sentinel could include model smoothing or neural pruning, as discussed in Section 3.4. Further investigations on backdoor attacks in DFL are encouraged.



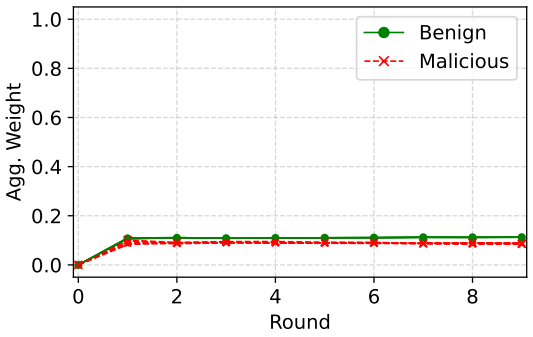
(a) Avg. Loss, PSR: 30%, FMNIST

(b) Avg. Loss, PSR: 30%, CIFAR10

Figure 6.17: The average bootstrap loss of models received from benign and malicious neighbors at participant n_0 under backdoor attack.



(a) Agg. Weight, PSR: 30%, FMNIST



(b) Agg. Weight, PSR: 30%, CIFAR10

Figure 6.18: The aggregation weights of models received from benign and malicious neighbors at participant n_0 under backdoor attack.

6.3.2 Threshold Analysis

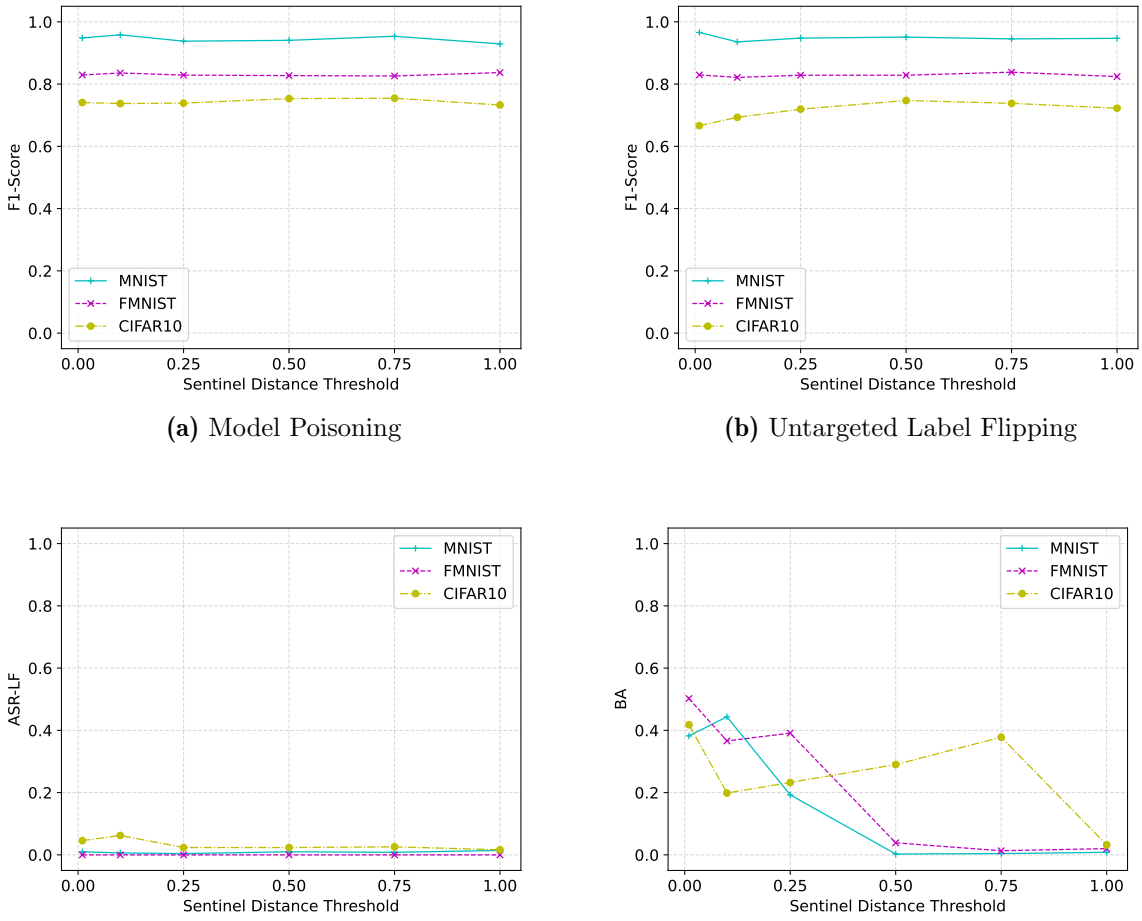
The loss distance threshold τ_L of Sentinel determines the extent to which a model will be aggregated. With the function *MapLossDistance* defined in Algorithm 4, each received model is assigned an aggregation weight in the range $[0, 1]$, which is further normalized. Models with a weight w_i lower than the predetermined threshold will be discarded. In general, setting the threshold τ_L to a small ϵ is recommended for numerical stability. For the experiments in Section 6.2, this threshold was fixed to $\tau_L = 0.5$. In the following, the impact of the loss threshold configuration is analyzed. Therefore, a set of experiments was run with Sentinel and all attacks from Section 4.1 with a PNR = 50 and a PSR = 100, for 10 rounds. A PSR is not required for model poisoning. The effectiveness of Sentinel was tested for the following loss threshold configurations: 0.01, 0.1, 0.25, 0.5, 0.75, 1. Essentially, with a threshold of $\tau = 1$, a node applying Sentinel only aggregates models with a bootstrap loss equal to or lower than its own. The success of the attack is measured by the corresponding metric defined in Section 4.1, whereby the mean result of all benign participants at round 10 is computed.

Arguably, Sentinel also uses a threshold for similarity filtering, specified by τ_S. However, as this threshold is used mainly to reduce the computational complexity of model aggregation, an analysis of τ_S is not within the scope of this section.

The results of these experiments are presented in Figure 6.19. The processed data is available on the repository[8]. For the three attacks model poisoning, untargeted label flipping and targeted label flipping, the threshold τ_L did not have a considerable impact on the defense effectiveness. In model poisoning, all nodes were able to reach the baseline F1-score for all datasets and any τ_L. For untargeted label flipping on CIFAR10, a drop in F1-score of approximately 7% from the baseline was observed with $\tau_L = 0.01$. Similar, for targeted label flipping on CIFAR10, a threshold of 0.01 resulted in a slight increase of ASR_{LF} to approx. 0.06.

[8] evaluation/metrics/sentinel_threshold

Additionally, it can be observed in Figure 6.19d that for MNIST and FMNIST, the BA decreases with an increasing threshold. This is not the case for CIFAR10, where the effectiveness of the backdoor attack seems to be highly variable, independent of the threshold. Only with a threshold $\tau_L > 0.75$, the backdoor could be mitigated. Nevertheless, the results highlight the need for an additional defense metric. While for model poisoning and label flipping, the cosine similarity and the bootstrap loss distance are reliable measures, they cannot effectively detect a poisoned model with a backdoor. It is not feasible to determine an appropriate loss threshold in advance in a realistic FL scenario. Hence, further investigations are required to improve Sentinel. A possible mitigation for backdoor attacks could be to set a threshold in relation to the current model performance and dynamically decrease the threshold as the FL process progresses.



(a) Model Poisoning

(b) Untargeted Label Flipping

(c) Targeted Label Flipping

(d) Backdoor

Figure 6.19: Different thresholds for all attacks with PNR = 50%, PSR = 100% (except model poisoning) on all datasets. The relevant performance metric is on the y-axis, and an increasing threshold τ_L is found on the x-axis.

6.3.3 Computational Advantages of SentinelGlobal

SentinelGlobal extends on Sentinel and applies a global trust perspective to decide whether a model should be discarded or evaluated, hence considered for aggregation. This concept of global trust is explained in Section 4.2.2. Briefly, the trust evaluation mechanism of SentinelGlobal retrieves the perspectives of the locally trusted neighbors on a target node at hand. A neighboring node is considered trusted if its model was involved in the aggregation of the previous round, *i.e.*, it received an aggregation weight $w_i > 0$. The aim of this global trust concept is to reduce the number of evaluations needed at each aggregation round, thus decrease the computational complexity of Sentinel.

To measure the effectiveness, the number of evaluations was recorded at each round for the same experiment configuration as in Section 6.2. For this evaluation, only the FMNIST dataset is considered, as the concept is independent on the dataset[9]. Table 6.9 summarizes the total number of models evaluated after the 10^{th} round for each combination of PSR and PNR at the benign participant n_0. With a total number of nodes |N|=10, the number of rounds R set to 10 and SentinelGlobal becoming activate at round $r_\alpha = 3$, there are always at least $|\text{N}| * r_\alpha = 30$ evaluations. From the round r_α, only the models considered trusted are evaluated and aggregated in the optimal scenario. Consequently, in the best case there are 44 evaluations for PNR = 80%, 65 for PNR = 50% and lastly 93 for PNR = 10%.

As illustrated in the results of Table 6.9, in 25 of the 39 experiments, SentinelGlobal was able to reduce the number of local evaluations to the optimum. In other cases, only a slight increase is observed. Thus, SentinelGlobal can effectively reduce the computational effort of Sentinel. This reduction depends on the number of participants in the network which are considered malicious.

Nevertheless, this computational advantage could not be clearly observed in the computational metrics listed in Section 6.2.6: although the number of evaluations was reduced by more than 50% for highly malicious environments, only a small amount of CPU and RAM decrease is notable in Figure 6.11. This means that the evaluation of Sentinel on a small bootstrap dataset and the model sizes considered in this work does not introduce a distinct overhead.

[9] evaluation/metrics/sentinelglobal

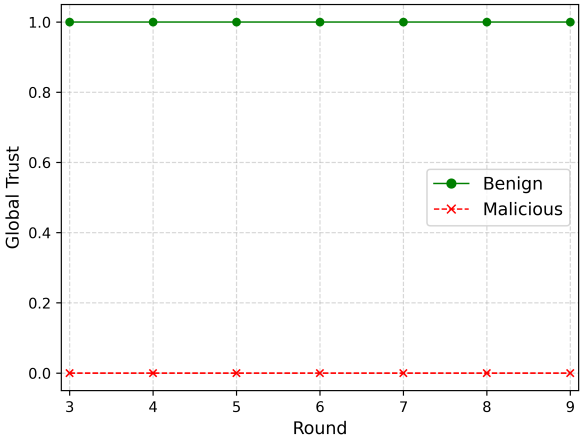
Table 6.9: The number of models evaluated with SentinelGlobal in use, for all attacks measured on benign participant n_0.

PNR	10				50				80			
PSR	0	30	50	100	0	30	50	100	0	30	50	100
Backdoor	-	93	100	93	-	65	65	65	-	45	53	44
Label Flipping Targeted	-	93	93	93	-	65	65	65	-	44	44	44
Label Flipping Untargeted	-	94	93	93	-	65	65	65	-	71	44	44
Model Poisoning	93	-	-	-	65	-	-	-	44	-	-	-

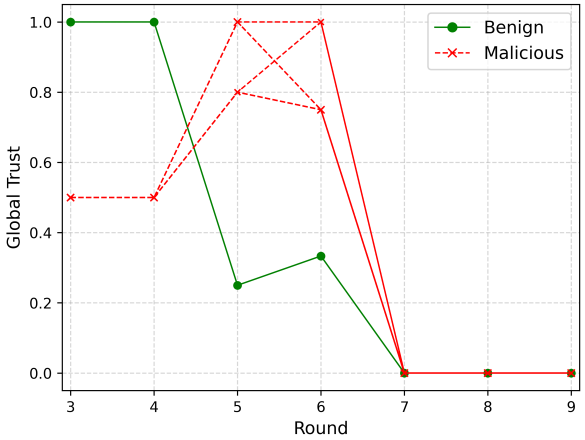
There are a few exceptions where SentinelGlobal was not effective: for untargeted label flipping and PNR = 80%, PSR = 30%, the number of evaluations is recognizably larger. Figure 6.20 illustrates the reason for these results. For a high PSR, the model evaluation mechanism of Sentinel could reliably distinguish between benign and malicious models. All attackers received a global trust score of 0 at each round and were consequently not evaluated. In contrast, SentinelGlobal was not reliable with a lower PSR of 30%: after round 5, all benign models received an average global trust score $\tilde{\Phi}_j < \tau_\chi = 0.5$. Instead, the malicious actors became trusted. Surprisingly, none of the neighbors was considered trusted from round 7 on, whether malicious or benign. This is the only scenario where SentinelGlobal classified benign neighbors as trusted and did not perform as expected.

As long as benign models are not incorrectly classified as malicious, the underlying aggregation mechanisms can still defend from poisoning attacks. This was observed for the backdoor attack, where all neighbors were considered trusted and consequently always evaluated. With a PSR = 50% and a PNR = 50%, SentinelGlobal correctly distinguished between malicious and benign neighbors. However, with a lower PSR of 50% and a single attacker, the adversary also became trusted, however with a lower $\tilde{\Phi}_j$ than benign neighbors.

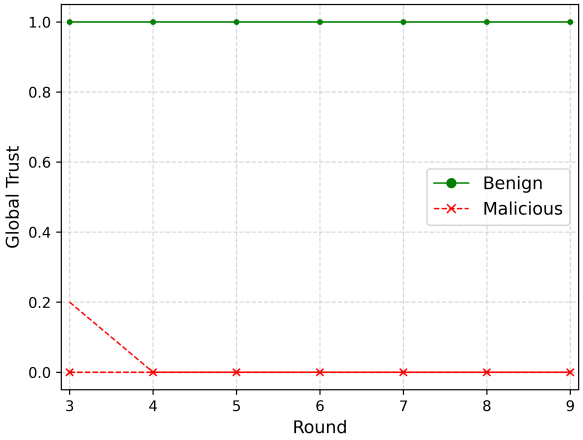
Hence, a more fine-grained trust concept might be required. Instead of binary trust scores, taking the aggregation weight as a trust score could be more reliable. Additionally, averaging the trust perspective over multiple or even all rounds could be advantageous. Another option could be to increase the influence of the local node within the global trust score computation. However, future improvements of Sentinel will consequently also improve the reliability of SentinelGlobal. In this context, it is also important to consider the attackers' perspective: with a low PSR, attacks such as label flipping are not effective. It can thus be expected that an attacker chooses a high PSR, whereas SentinelGlobal demonstrated a high reliability. On another node, network topologies other than a fully connected network have not yet been investigated. A lower node connectivity within the FL process may have an impact on the performance of SentinelGlobal.



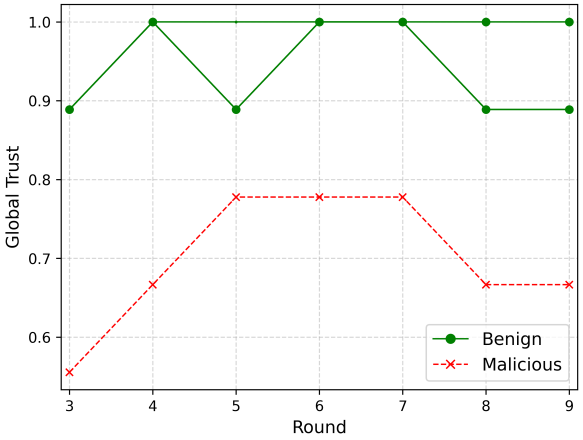
(a) Targeted Label Flipping, PNR: 80%, PSR: 100%



(b) Targeted Label Flipping, PNR: 80%, PSR: 30%



(c) Backdoor, PNR: 50%, PSR: 100%



(d) Backdoor, PNR: 10%, PSR: 50%

Figure 6.20: The SentinelGlobal trust scores recorded on participant n_0 on FMNIST under targeted label flipping and backdoor attack.

Chapter 7

Summary and Conclusions

In this work, two sophisticated defense strategies have been proposed: Sentinel and SentinelGlobal. Sentinel applies a multi-level defense protocol composed of similarity filtering, bootstrap validation and model normalization to mitigate poisoning attacks. By taking advantage of the local data availability in DFL, Sentinel demonstrates the transferability of promising defense strategies proposed for CFL. SentinelGlobal extends on the concepts of Sentinel with a global trust framework to reduce the threat of adversaries and decrease the computational complexity of Sentinel. Extensive evaluations on the datasets MNIST, FMNIST and CIFAR10 with various attack configurations have demonstrated their effectiveness. Compared to other state-of-the-art aggregation algorithms, the aggregation protocols proposed in this work reliably defended against poisoning attacks, especially in highly malicious environments. Furthermore, this work demonstrated the usability of Fedstellar for more comparable security benchmarks in DFL. Individual customizations have outlined its adaptability to the needs of specific research goals.

Specifically targeted sample poisoning attacks, *i.e.*, backdoors, have been found to be difficult to defend. For increasingly complex learning tasks, such as training a CNN on CIFAR10, not even Sentinel and SentinelGlobal were able to effectively defend against this type of attack without relying on an appropriately configured threshold. Possible improvements to Sentinel could be to use a more fine-grained similarity filtering approach and adaptive loss thresholds in relation to the local model performance. Additional defense layers that incorporate weak differential privacy, neuron pruning, or novel strategies against backdoor attacks may be promising components to further enhance the effectiveness of Sentinel. Additionally, this work only investigated simple, artificial backdoors. Other backdoor strategies, such as distributed or semantic backdoor attacks, are yet to be studied.

Investigations on the computational resource usage have demonstrated that SentinelGlobal can effectively reduce the number of model evaluations required by Sentinel and consequently decrease its computation complexity. However, selected experiments have demonstrated that the reliability of SentinelGlobal is highly dependent on the performance of the underlying aggregation mechanisms. To establish a robust binary trust score, a defense protocol is required to confidently distinguish between malicious and benign. Future improvements to Sentinel consequently enhance the trust evaluation of

SentinelGlobal. Despite that, supplemental improvements, such as a more fine-grained trust protocol that incorporates temporal aspects, could also be the focus of future work. Concretely, this could involve using the aggregation weight as the local trust perspective on a node over multiple rounds.

Furthermore, all conducted experiments used a fully connected network of ten nodes. It is thus important to consider that the investigations of this work evaluated the worst case scenario in DFL. In a fully connected network, malicious actors are given the opportunity to attack every participant in the network. With scenarios where more than the majority is malicious, defending against poisoning attacks becomes a highly difficult task. Other network topologies such as ring topologies or random graphs with or without clustered components may reveal interesting results and reflect more realistic applications of DFL. On another note, only datasets in an IID setting were analyzed. With future findings to improve the convergence of DFL in non-IID scenarios, the defense techniques should also be evaluated in more heterogeneous experiments.

Bibliography

[1] M. Alazab, S. P. Rm, P. M, P. K. R. Maddikunta, T. R. Gadekallu, and Q.-V. Pham, "Federated Learning for Cybersecurity: Concepts, Challenges, and Future Directions", en, *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3501–3509, May 2022, ISSN: 1551-3203, 1941-0050. DOI: 10.1109/TII.2021.3119038. [Online]. Available: https://ieeexplore.ieee.org/document/9566732/ (visited on 02/21/2023).

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, *Communication-Efficient Learning of Deep Networks from Decentralized Data*, arXiv:1602.05629 [cs], Feb. 2016. DOI: 10.48550/arXiv.1602.05629. [Online]. Available: http://arxiv.org/abs/1602.05629 (visited on 02/20/2023).

[3] 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, May 2020, arXiv:1908.07873 [cs, stat], ISSN: 1053-5888, 1558-0792. DOI: 10.1109/MSP.2020.2975749. [Online]. Available: http://arxiv.org/abs/1908.07873 (visited on 02/17/2023).

[4] L. He, A. Bian, and M. Jaggi, "COLA: Decentralized Linear Learning", in *Advances in Neural Information Processing Systems*, vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/hash/05a70454516ecd9194c293b0e415777f-Abstract.html (visited on 08/02/2023).

[5] N. Rodríguez-Barroso, D. Jiménez-López, M. V. Luzón, F. Herrera, and E. Martínez-Cámara, "Survey on federated learning threats: Concepts, taxonomy on attacks and defences, experimental study and challenges", en, *Information Fusion*, vol. 90, pp. 148–173, Feb. 2023, ISSN: 15662535. DOI: 10.1016/j.inffus.2022.09.011. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1566253522001439 (visited on 03/08/2023).

[6] 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, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2023.3238823.

[7] 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*, arXiv:2211.08413 [cs], Nov. 2022. DOI: 10.48550/arXiv.2211.08413. [Online]. Available: http://arxiv.org/abs/2211.08413 (visited on 02/13/2023).

[8] T. Yang, G. Andrew, H. Eichner, *et al.*, *Applied Federated Learning: Improving Google Keyboard Query Suggestions*, arXiv:1812.02903 [cs, stat], Dec. 2018. DOI: 10.48550/arXiv.1812.02903. [Online]. Available: http://arxiv.org/abs/1812.02903 (visited on 02/20/2023).

[9] P. Bellavista, L. Foschini, and A. Mora, "Decentralised Learning in Federated Deployment Environments: A System-Level Survey", *ACM Computing Surveys*, vol. 54, no. 1, 15:1–15:38, Feb. 2021, ISSN: 0360-0300. DOI: 10.1145/3429252. [Online]. Available: https://doi.org/10.1145/3429252 (visited on 02/17/2023).

[10] J. C. Jiang, B. Kantarci, S. Oktug, and T. Soyata, "Federated Learning in Smart City Sensing: Challenges and Opportunities", en, *Sensors*, vol. 20, no. 21, p. 6230, Jan. 2020, Number: 21 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1424-8220. DOI: 10.3390/s20216230. [Online]. Available: https://www.mdpi.com/1424-8220/20/21/6230 (visited on 07/27/2023).

[11] C. Xie, S. Koyejo, and I. Gupta, *Zeno++: Robust Fully Asynchronous SGD*, arXiv:1903.07020 [cs, stat], May 2021. DOI: 10.48550/arXiv.1903.07020. [Online]. Available: http://arxiv.org/abs/1903.07020 (visited on 03/08/2023).

[12] O. Friha, M. A. Ferrag, M. Benbouzid, T. Berghout, B. Kantarci, and K.-K. R. Choo, "2DF-IDS: Decentralized and differentially private federated learning-based intrusion detection system for industrial IoT", en, *Computers & Security*, vol. 127, p. 103 097, Apr. 2023, ISSN: 0167-4048. DOI: 10.1016/j.cose.2023.103097. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016740482300007X (visited on 03/15/2023).

[13] A. Brecko, E. Kajati, J. Koziorek, and I. Zolotova, "Federated Learning for Edge Computing: A Survey", en, *Applied Sciences*, vol. 12, no. 18, p. 9124, Sep. 2022, ISSN: 2076-3417. DOI: 10.3390/app12189124. [Online]. Available: https://www.mdpi.com/2076-3417/12/18/9124 (visited on 02/24/2023).

[14] 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, Type: Review tex.publication_stage: Final tex.source: Scopus. DOI: 10.1561/2200000083. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85108873867&doi=10.1561%2f2200000083&partnerID=40&md5=303cc9306ba3e493f0e6404346ece2d5.

[15] P. R. Silva, J. Vinagre, and J. Gama, "Towards federated learning: An overview of methods and applications", en, *WIREs Data Mining and Knowledge Discovery*, Jan. 2023, ISSN: 1942-4787, 1942-4795. DOI: 10.1002/widm.1486. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/widm.1486 (visited on 02/21/2023).

[16] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, *How To Backdoor Federated Learning*, arXiv:1807.00459 [cs], Aug. 2019. DOI: 10.48550/arXiv.1807.00459. [Online]. Available: http://arxiv.org/abs/1807.00459 (visited on 03/14/2023).

[17] V. Shejwalkar, A. Houmansadr, P. Kairouz, and D. Ramage, "Back to the Drawing Board: A Critical Evaluation of Poisoning Attacks on Production Federated Learning", en, in *2022 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA: IEEE, May 2022, pp. 1354–1371, ISBN: 978-1-66541-316-9. DOI: 10.1109/SP46214.2022.9833647. [Online]. Available: https://ieeexplore.ieee.org/document/9833647/ (visited on 03/02/2023).

[18] A. K. Nair, E. D. Raj, and J. Sahoo, "A robust analysis of adversarial attacks on federated learning environments", en, *Computer Standards & Interfaces*, vol. 86, p. 103 723, Aug. 2023, ISSN: 09205489. DOI: 10.1016/j.csi.2023.103723. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0920548923000041 (visited on 03/13/2023).

[19] M. Fang, X. Cao, J. Jia, and N. Z. Gong, *Local Model Poisoning Attacks to Byzantine-Robust Federated Learning*, arXiv:1911.11815 [cs], Nov. 2021. DOI: 10.48550/arXiv.1911.11815. [Online]. Available: http://arxiv.org/abs/1911.11815 (visited on 03/14/2023).

[20] M. S. Jere, T. Farnan, and F. Koushanfar, "A Taxonomy of Attacks on Federated Learning", en, *IEEE Security & Privacy*, vol. 19, no. 2, pp. 20–28, Mar. 2021, ISSN: 1540-7993, 1558-4046. DOI: 10.1109/MSEC.2020.3039941. [Online]. Available: https://ieeexplore.ieee.org/document/9308910/ (visited on 03/10/2023).

[21] L. Lyu, H. Yu, and Q. Yang, *Threats to Federated Learning: A Survey*, arXiv:2003.02133 [cs, stat], Mar. 2020. DOI: 10.48550/arXiv.2003.02133. [Online]. Available: http://arxiv.org/abs/2003.02133 (visited on 03/13/2023).

[22] Z. Tian, L. Cui, J. Liang, and S. Yu, "A Comprehensive Survey on Poisoning Attacks and Countermeasures in Machine Learning", en, *ACM Computing Surveys*, vol. 55, no. 8, pp. 1–35, Aug. 2023, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3551636. [Online]. Available: https://dl.acm.org/doi/10.1145/3551636 (visited on 07/09/2023).

[23] P. Rieger, T. D. Nguyen, M. Miettinen, and A.-R. Sadeghi, "DeepSight: Mitigating Backdoor Attacks in Federated Learning Through Deep Model Inspection", en, in *Proceedings 2022 Network and Distributed System Security Symposium*, arXiv:2201.00763 [cs], 2022. DOI: 10.14722/ndss.2022.23156. [Online]. Available: http://arxiv.org/abs/2201.00763 (visited on 03/15/2023).

[24] M. S. Ozdayi, M. Kantarcioglu, and Y. R. Gel, "Defending against Backdoors in Federated Learning with Robust Learning Rate", en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, pp. 9268–9276, May 2021, ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v35i10.17118. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17118 (visited on 03/08/2023).

[25] M. Baruch, G. Baruch, and Y. Goldberg, *A Little Is Enough: Circumventing Defenses For Distributed Learning*, en, arXiv:1902.06156 [cs, stat], Feb. 2019. [Online]. Available: http://arxiv.org/abs/1902.06156 (visited on 03/07/2023).

[26] L. Li, W. Xu, T. Chen, G. B. Giannakis, and Q. Ling, "RSA: Byzantine-Robust Stochastic Aggregation Methods for Distributed Learning from Heterogeneous Datasets", en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 1544–1551, Jul. 2019, ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v33i01.33011544. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/3968 (visited on 04/11/2023).

[27] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "The Limitations of Federated Learning in Sybil Settings", en, 2020.

[28] T. D. Nguyen, T. Nguyen, P. L. Nguyen, H. H. Pham, K. Doan, and K.-S. Wong, *Backdoor Attacks and Defenses in Federated Learning: Survey, Challenges and Future Research Directions*, arXiv:2303.02213 [cs], Mar. 2023. [Online]. Available: http://arxiv.org/abs/2303.02213 (visited on 03/14/2023).

[29] Y. Zhang, D. Zeng, J. Luo, Z. Xu, and I. King, *A Survey of Trustworthy Federated Learning with Perspectives on Security, Robustness, and Privacy*, arXiv:2302.10637 [cs], Feb. 2023. DOI: 10.48550/arXiv.2302.10637. [Online]. Available: http://arxiv.org/abs/2302.10637 (visited on 03/16/2023).

[30] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The Hidden Vulnerability of Distributed Learning in Byzantium", en, 2018.

[31] S. Guo, T. Zhang, H. Yu, *et al.*, *Byzantine-resilient Decentralized Stochastic Gradient Descent*, en, arXiv:2002.08569 [cs], Oct. 2021. [Online]. Available: http://arxiv.org/abs/2002.08569 (visited on 05/18/2023).

[32] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping", en, in *Proceedings 2021 Network and Distributed System Security Symposium*, Virtual: Internet Society, 2021, ISBN: 978-1-891562-66-2. DOI: 10.14722/ndss.2021.24434. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_6C-2_24434_paper.pdf (visited on 03/15/2023).

[33] J. Yin, X. Cui, and K. Li, "A Reputation-Based Resilient and Recoverable P2P Botnet", in *2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*, Jun. 2017, pp. 275–282. DOI: 10.1109/DSC.2017.20.

[34] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-Robust distributed learning: Towards optimal statistical rates", in *Proceedings of the 35th international conference on machine learning*, J. Dy and A. Krause, Eds., ser. Proceedings of machine learning research, vol. 80, PMLR, Jul. 2018, pp. 5650–5659. [Online]. Available: https://proceedings.mlr.press/v80/yin18a.html.

[35] K. Pillutla, S. M. Kakade, and Z. Harchaoui, *Robust Aggregation for Federated Learning*, arXiv:1912.13445 [cs, stat], Jan. 2022. DOI: 10.48550/arXiv.1912.13445. [Online]. Available: http://arxiv.org/abs/1912.13445 (visited on 03/07/2023).

[36] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent", in *Proceedings of the 31st international conference on neural information processing systems*, ser. NIPS'17, Number of pages: 11 Place: Long Beach, California, USA, Red Hook, NY, USA: Curran Associates Inc., 2017, pp. 118–128, ISBN: 978-1-5108-6096-4.

[37] S. Li, Y. Cheng, W. Wang, Y. Liu, and T. Chen, *Learning to Detect Malicious Clients for Robust Federated Learning*, en, arXiv:2002.00211 [cs, stat], Feb. 2020. [Online]. Available: http://arxiv.org/abs/2002.00211 (visited on 03/20/2023).

[38] Y. Wang, D.-H. Zhai, Y. He, and Y. Xia, "An adaptive robust defending algorithm against backdoor attacks in federated learning", en, *Future Generation Computer Systems*, vol. 143, pp. 118–131, Jun. 2023, ISSN: 0167739X. DOI: 10.1016/j.future.2023.01.026. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X2300033X (visited on 02/24/2023).

[39] Z. Sun, P. Kairouz, A. T. Suresh, and H. B. McMahan, *Can You Really Backdoor Federated Learning?*, en, arXiv:1911.07963 [cs, stat], Dec. 2019. [Online]. Available: http://arxiv.org/abs/1911.07963 (visited on 02/28/2023).

[40] C. Fung, C. J. M. Yoon, and I. Beschastnikh, *Mitigating Sybils in Federated Learning Poisoning*, arXiv:1808.04866 [cs, stat], Jul. 2020. DOI: 10.48550/arXiv.1808.04866. [Online]. Available: http://arxiv.org/abs/1808.04866 (visited on 02/17/2023).

[41] C. Fang, Z. Yang, and W. U. Bajwa, *BRIDGE: Byzantine-resilient Decentralized Gradient Descent*, en, arXiv:1908.08098 [cs, eess, stat], Jun. 2022. [Online]. Available: http://arxiv.org/abs/1908.08098 (visited on 03/16/2023).

[42] L. Muñoz-González, K. T. Co, and E. C. Lupu, *Byzantine-Robust Federated Machine Learning through Adaptive Model Averaging*, en, arXiv:1909.05125 [cs, stat], Sep. 2019. [Online]. Available: http://arxiv.org/abs/1909.05125 (visited on 02/20/2023).

[43] T. D. Nguyen, P. Rieger, H. Chen, *et al.*, *FLAME: Taming Backdoors in Federated Learning*, Report Number: 025, 2021. [Online]. Available: https://eprint.iacr.org/2021/025 (visited on 03/15/2023).

[44] V. Shejwalkar and A. Houmansadr, "Manipulating the Byzantine: Optimizing Model Poisoning Attacks and Defenses for Federated Learning", en, in *Proceedings 2021 Network and Distributed System Security Symposium*, Virtual: Internet Society, 2021, ISBN: 978-1-891562-66-2. DOI: 10.14722/ndss.2021.24498. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/ndss2021_6C-3_24498_paper.pdf (visited on 03/04/2023).

[45] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, *signSGD: Compressed Optimisation for Non-Convex Problems*, arXiv:1802.04434 [cs, math], Aug. 2018. DOI: 10.48550/arXiv.1802.04434. [Online]. Available: http://arxiv.org/abs/1802.04434 (visited on 03/08/2023).

[46] Y. Zhao, J. Chen, J. Zhang, D. Wu, J. Teng, and S. Yu, "PDGAN: A Novel Poisoning Defense Method in Federated Learning Using Generative Adversarial Network", en, in *Algorithms and Architectures for Parallel Processing*, S. Wen, A. Zomaya, and L. T. Yang, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2020, pp. 595–609, ISBN: 978-3-030-38991-8. DOI: 10.1007/978-3-030-38991-8_39.

[47] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, *FLDetector: Defending Federated Learning Against Model Poisoning Attacks via Detecting Malicious Clients*, arXiv:2207.09209 [cs], Oct. 2022. DOI: 10.48550/arXiv.2207.09209. [Online]. Available: http://arxiv.org/abs/2207.09209 (visited on 03/14/2023).

[48] D. Cao, S. Chang, Z. Lin, G. Liu, and D. Sun, "Understanding Distributed Poisoning Attack in Federated Learning", in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, ISSN: 1521-9097, Dec. 2019, pp. 233–239. DOI: 10.1109/ICPADS47876.2019.00042.

[49] B. Zhao, P. Sun, T. Wang, and K. Jiang, "FedInv: Byzantine-Robust Federated Learning by Inversing Local Model Updates", en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 8, pp. 9171–9179, Jun. 2022, Number: 8, ISSN: 2374-3468. DOI: 10.1609/aaai.v36i8.20903. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/20903 (visited on 03/23/2023).

[50] A. Gholami, N. Torkzaban, and J. S. Baras, "Trusted Decentralized Federated Learning", en, in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA: IEEE, Jan. 2022, pp. 1–6, ISBN: 978-1-66543-161-3. DOI: 10.1109/CCNC49033.2022.9700624. [Online]. Available: https://ieeexplore.ieee.org/document/9700624/ (visited on 02/21/2023).

[51] S. P. Karimireddy, L. He, and M. Jaggi, *Learning from History for Byzantine Robust Optimization*, arXiv:2012.10333 [cs, math, stat], Jun. 2021. [Online]. Available: http://arxiv.org/abs/2012.10333 (visited on 03/28/2023).

[52] L. He, S. P. Karimireddy, and M. Jaggi, *Byzantine-Robust Decentralized Learning via Self-Centered Clipping*, en, arXiv:2202.01545 [cs, math, stat], Feb. 2022. [Online]. Available: http://arxiv.org/abs/2202.01545 (visited on 02/24/2023).

[53] C. Wu, X. Yang, S. Zhu, and P. Mitra, *Mitigating Backdoor Attacks in Federated Learning*, en, arXiv:2011.01767 [cs], Jan. 2021. [Online]. Available: http://arxiv.org/abs/2011.01767 (visited on 03/04/2023).

[54] C. Xie, M. Chen, P.-Y. Chen, and B. Li, *CRFL: Certifiably Robust Federated Learning against Backdoor Attacks*, en, arXiv:2106.08283 [cs], Jun. 2021. [Online]. Available: http://arxiv.org/abs/2106.08283 (visited on 03/04/2023).

[55] E. T. M. Beltrán, Á. L. P. Gómez, C. Feng, *et al.*, *Fedstellar: A Platform for Decentralized Federated Learning*, arXiv:2306.09750 [cs], Jun. 2023. [Online]. Available: http://arxiv.org/abs/2306.09750 (visited on 07/09/2023).

[56] Y. LeCun and C. Cortes, "MNIST handwritten digit database", 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/ (visited on 01/14/2016).

[57] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms*, arXiv:1708.07747 [cs, stat], Sep. 2017. DOI: 10.48550/arXiv.1708.07747. [Online]. Available: http://arxiv.org/abs/1708.07747 (visited on 07/11/2023).

[58] A. Krizhevsky, "Learning multiple layers of features from tiny images", Tech. Rep., 2009.

[59] A. G. Howard, M. Zhu, B. Chen, *et al.*, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, arXiv:1704.04861 [cs], Apr. 2017. [Online]. Available: http://arxiv.org/abs/1704.04861 (visited on 07/11/2023).

Abbreviations

ML Machine Learning

SGD Stochastic Gradient Descent

FL Federated Learning

CFL Centralized Federated Learning

DFL Decentralized Federated Learning

IID independent and identically distributed

HFL Horizontal Federated Learning

VFL Vertical Federated Learning

SEM Standard Error Mean

PNR Poisoned Node Ratio

PSR Poisoned Sample Ratio

NR Noise Ratio

ASR Attacker Success Rate

List of Figures

List of Tables

List of Algorithms

List of Listings

Appendix A

Installation Guidelines

Technical instructions on running the adapted version of Fedstellar and reproducing the experiments in this work can be retrieved from the corresponding GitHub repository: https://github.com/janousy/fedstellar-sync.

Appendix B

Evaluation

B.1 Fedstellar Configuration

Listing 4 Fedstellar Scenario Parameters

```
{
  "BLOCK_SIZE": 4096,
  "NODE_TIMEOUT": 20,
  "VOTE_TIMEOUT": 60,
  "AGGREGATION_TIMEOUT": 60,
  "HEARTBEAT_PERIOD": 4,
  "HEARTBEATER_REFRESH_NEIGHBORS_BY_PERIOD": 4,
  "WAIT_HEARTBEATS_CONVERGENCE": 10,
  "TRAIN_SET_SIZE": 10,
  "TRAIN_SET_CONNECT_TIMEOUT": 100,
  "AMOUNT_LAST_MESSAGES_SAVED": 100,
  "GOSSIP_MESSAGES_FREC": 100,
  "GOSSIP_MESSAGES_PER_ROUND": 100,
  "GOSSIP_EXIT_ON_X_EQUAL_ROUNDS": 20,
  "GOSSIP_MODELS_FREC": 1,
  "GOSSIP_MODELS_PER_ROUND": 5,
  "ROUND_PROCEED_TIMEOUT": 180
}
```