RADAR: Model Quality Assessment for Reputation-aware Collaborative Federated Learning

Redacted for double-blind
Redacted Lab
Redacted
redacted@redacted.com

Redacted for double-blind
Redacted Lab
Redacted
redacted@redacted.com

Redacted for double-blind
Redacted Lab
Redacted
redacted@redacted.com

Redacted for double-blind
Redacted Lab
Redacted
redacted@redacted.com

***Abstract*—Cross-silo federated learning (CS-FL) is a distributed learning setting which allows an identified set of organizations to collaboratively train a single global model. Since CS-FL use cases are often heterogeneous, it may be more appropriate to dynamically provide different models to more homogeneous sub-federations. In addition, such systems can be undermined by contributions of poor quality, making negligent or even malicious participants critical to consider. However, distinguishing such participants in a heterogeneous context is especially difficult.**

We present `RADAR`, a novel architecture for CS-FL able to assess the quality of the participants' contributions, regardless of data similarity. `RADAR` leverages client-side evaluation to directly collect feedbacks from the participants. The same evaluations allow grouping participants according to their perceived similarity and weighting the model aggregation based on their reputation. To evaluate our approach on concrete experiments, we implement a collaborative intrusion detection system (CIDS) scenario and test our architecture in various data-quality settings using label-flipping. Our results confirm that combining clustering and a reputation system succeeds in detecting a wide range of Byzantine behaviors, including colluding attackers, which highlights `RADAR`'s versatility.

***Index Terms*—federated learning, intrusion detection, Byzantine, cross-evaluation, similarity, clustering, reputation systems, trust, heterogeneity,**

I. Introduction

Collaborative machine learning (ML) enables multiple organizations to train a common model that benefits from each other's experience. Despite its advantages, it has received a lukewarm welcome from user communities, mainly because of the risks induced by data sharing. The recent advances [1] in federated learning (FL) promise to solve such issues, allowing participants to collaboratively train a global model without sharing their local data [2]. Specifically, organizations involved in collaborative ML can leverage horizontal federated learning (HFL) (*i.e.* same features, but different samples), to share their observations with other participants while keeping their locally collected network data private [3]. This configuration is referred to as cross-silo federated learning (CS-FL), as the participants act as siloed data-sources. Participants in CS-FL are typically fewer, stateful, and long-lived, but also can own highly heterogeneous data, depending on their local use cases and objectives [1].

A practical use case for CS-FL is collaborative intrusion detection systems (CIDSs) [3], as sharing network data can expose information about the inner workings of information systems. Moreover, different organizations might exhibit substantial differences in their information systems, such as hosted services or used protocols. This can lead to significant variations in model updates, as each organization trains its model on its local network traffic. Furthermore, as collaborative systems are especially sensitive to input quality, Byzantine failures must be considered. Indeed, honest participants can negatively contribute to the aggregation by training their model on data of poor quality or by being unaware of attacks present in their network. Malicious participants in a CIDS could even poison their contributions to impact the convergence of the global model, or introduce weaknesses that could be exploited afterward. In this heterogeneous context, it is very difficult to distinguish a faulty or malicious contribution from a legitimate one originating from a different type of infrastructure.

Approaches that assess model quality [4] or mitigate poisoning [5], [6] in homogeneous distributions typically compare or evaluate a model using a single source of truth. Building such a single source of truth, however, is inadequate in heterogeneous contexts due to the differences between participants. Assuming that all contributions are therefore different, some approaches detect colluding attackers based on their similarity [7], [8]. Nevertheless, these approaches fail to detect an isolated, yet potent, attacker.

In this paper, we present `RADAR`, an architecture for CS-FL guarantying high-quality model aggregation, regardless of the data homogeneity. `RADAR` relies on three main ingredients: *i)* a modified FL workflow, where each participant uses its local dataset to evaluate the other participants' models, between the training and aggregation steps; *ii)* a clustering algorithm leveraging the participants' perceived similarity to aggregate group-specific global models; and *iii)* a reputation system that weights the participants' contributions based on their past interactions.

We evaluate the performance of `RADAR` in a realistic CIDS use case, using four network flow datasets with standardized features, representing different environments, and model various Byzantine behavior using label-flipping. We also compare our approach to existing strategies [2], [7], and conclude that `RADAR` can detect Byzantines contributions under most scenarios, from noisy labels to colluding poisoning attacks.

To summarize, our contributions are threefold:

(1) we present RADAR, an architectural framework to protect FL strategies using clustering and reputation-aware aggregation; validated by extensive evaluation against relevant baselines;
(2) we show that evaluation metrics (such as accuracy, F1-score, or loss) can be used to effectively assess similarity between FL participants, and as an input to clustering and reputation algorithms;
(3) we validate that combining reputation and clustering successfully addresses the problem of contribution quality assessment in heterogeneous settings.

The rest of this paper is organized as follows. Section II defines the addressed problem and introduces our threat model. In Section III we present related works and their remaining limitations. Section IV presents RADAR's design and core components, before discussing implementation and evaluation methodology in Section V. We provide an extensive evaluation of this approach in Section VI. We discuss our findings in Section VII before concluding and laying out future works in Section VIII.

II. Preliminaries and Problem Statement

A. CIDS using Federated Learning

We consider a typical FL scenario where a central server S is tasked with aggregating the model updates w_i^r of n participants $p_i, i \in [\![1, n]\!]$ at each round r. Participants are entities that oversee an organization's network, which makes them highly available and interested. This is analogous to CS-FL settings [1], where there are also few participants with consequent quantities of data, and significant computing capabilities.

Additionally, we set the proportion of selected clients to 1.0 to use FL as a collaborative framework, where all clients contribute to the global model and get updates at each round. We denote respectively by P and W^r the sets of all participants and all local parameters. Model architecture and hyperparameters are the same among participants, but each owns a local dataset d_i that is not shared with the others.

To work on a realistic application, we implement a network-based intrusion detection system (NIDS) use case, where d_i is composed of labeled network flows, categorized in two classes: *benign* and *malicious*. Because organizations in CIDS may have different network configurations [9], the distribution of each local dataset d_i can vary considerably, independently of the associated labels. This is typically the case in CS-FL and is referred to as *not independent or identically distributed (non-IID)* settings. However, the CIDS use case implies that similarities can exist between participants, for instance between organizations operating in the same sector or having similar network infrastructure. This particular setting can be described as *practical non-IID*, as opposed to the *pathological non-IID* settings, where all participants have unique and highly different data-distributions [10].

At each round r, and using their local dataset d_i, each participant trains a parametric model—*e.g.*, deep neural network (DNN)—on a binary classification task, *i.e.* predicting each sample's labels. This amounts to minimizing a loss function $\mathcal{L}(w_i^r, \vec{x_j}, \vec{y_j}), j \in [\![1, |d_i|]\!]$, where $\vec{x_j}$ and $\vec{y_j}$ refer to the sample and its label, respectively. To that end, they use a stochastic gradient descent (SGD)-based optimizer to compute the gradients $\nabla\mathcal{L}(w_i^r, \vec{x_j}, \vec{y_j})$ and update their new model as

$$w_i^{r+1} \leftarrow w_i^r - \eta\nabla\mathcal{L}(w_i^r, \vec{x_j}, \vec{y_j}), \qquad (1)$$

where the η is the learning rate. The server then computes the new global model $\overline{w}^r$ as a function of the local models $\{w_i^r \mid i \in [\![1, n]\!]\}$, akin to FedAvg [2].

B. Low-quality Contributions

In FL, the quality of the global model is directly impacted by the quality of the participants' contributions. In a intrusion detection system (IDS) context, the poor quality of a ML model can be induced by some choices in terms of architecture, hyperparameters, or optimizer—all fixed by the server, but also by the quality of the training data. Multiple factors can affect the quality of local training data [11], such as: (1) *Label noise*—samples associated with the wrong labels; (2) *Class imbalance*—differences in terms of class representation in the dataset; or (3) *Data heterogeneity*—the variations between samples of the same class.

Similar to existing works on data-quality [12], [13], we focus on label noise, which can have significant consequences on the global model's performance, depending on the proportion of mislabeled samples. In a CIDS, label noise can unknowingly be introduced by the participants, either due to misconfigurations or to the presence of compromised devices. We consider two types of label noise: *missed intrusions* and *misclassification*.

a) *Missed intrusions* occur when a malicious sample is mislabeled as benign, leading to a false negative. Participants in CIDSs label the attacks they are aware of, but some might have been unnoticed.
b) A *misclassification* is the random mislabeling of a sample. This can be due to a lack of knowledge or to a misconfiguration.

Such participants are referred to as *honest-but-neglectful*. Because these errors are assumed to be unintentional, the proportion of *misclassified* samples is expected to be low. However, the concept of *missed intrusions* implies that the participants are not aware of an entire attack, which can represent a significant proportion of their dataset.

C. Data Poisoning Attacks

In addition to accidental low-quality contributions, some participants might deliberately upload model updates that would negatively impact the performance of the global model.

We refer to them as *malicious participants*. Malicious behavior can be modeled by poisoning attacks, in which an attacker would alter his contribution to impact the performance of the global model. The literature distinguishes two classes of poisoning attacks: *data poisoning* and *model poisoning*. In the former, an attacker can tamper with the training data set, but otherwise faithfully executes its process [7], [8]. In the latter, the attacker directly modifies the model updates sent to the server [14]–[16].

In this paper, we focus on data poisoning attacks, as they are fairly accessible to any type of attacker. Specifically, we study label-flipping attacks, as it can effectively model both legitimate participants whose training data has been altered, and malicious participants who deliberately modify their training data. These attacks can further be separated into two categories. With *targeted poisoning*, an attacker modifies the behavior of the global model when it is subjected to a specific class [7], whereas with *untargeted poisoning*, the attacker tries to impact the model performance uniformly [6]. We exclude backdoor attacks [17] for this study, as they imply an attacker is purposely crafting poisoned samples.

Attackers' Knowledge: We consider *gray-box* adversaries, meaning they have the same knowledge as legitimate clients. Such information includes the last global models, the used hyperparameters, loss function, and model architecture.

Attackers' Objective: An attacker can choose the appropriate attack depending on his objective. With targeted poisoning, attackers aim at making a specific type of attack invisible to the NIDS. With untargeted attacks, on the other hand, they aim at maximizing the misclassification rate to jeopardize the NIDS performance.

Attackers' Capabilities: We consider multiple *noisiness* scenarios, *i.e.* the proportion of the attackers' training set that is poisoned at each round. The noisiness of an attacker over time (*i.e.* rounds) represents its behavior. While we consider the clients to remain the same in a collaborative NIDS environment, an attack can be triggered at anytime with any noisiness. Additionally, malicious actors can act alone or be involved in coordinated attacks. FoolsGold [7] focuses on Sybil attacks, a specific case of colluding attackers controlled by a single entity. We prefer the more generic term of *colluding attackers*, sharing common goal and means. Their number and proportion among benign clients can vary from a single *lone* attacker to them being a majority in the system.

D. Problem Formalization

Based on the previous assumptions, we consider that participants might upload model updates that would negatively impact the performance of the global model, deliberately or not. Multiple forms of such actors can exist: external actors altering legitimate clients' data (*i.e. compromised*), clients whose local training sets are of poor quality (*i.e. honest-but-neglectful*), or clients modifying their own local data on purpose (*i.e. malicious*). We refer to them as *Byzantine participants* or simply *Byzantines* in the remaining of this paper.

We further consider that the server can be trusted to perform the aggregation faithfully, and that FL guaranties the confidentiality of the local datasets. Attacking the server is out of the scope of this contribution. Consequently, we aim at weighting or discarding the participants' contributions based on their quality to guaranty the performance of the aggregated model.

Problem (Quality Assessment in Heterogeneous Settings)**.** *For n participants p_i and their local datasets d_i of unknown similarity, each participant uploads a model update w_i^r at each round r. Given $P = \{p_1, p_2, \ldots, p_n\}$ and $W^r = \{w_1^r, w_2^r, \ldots, w_n^r\}$, how can one assess the quality of each participant's contribution w_i^r without making assumptions on the data distribution across the datasets d_i?*

III. Related Work

A. Byzantine-resilient Federated Learning

The reliability of a submitted local model can be assessed in several ways, whether it is used to detect *honest-but-neglectful* or explicitly *malicious* participants. Some approaches use evaluation to validate submitted models against a centralized dataset [6], or against randomly selected distributed datasets [4] if they are representative of each other—which is the case with independent and identically distributed (IID) data partitioning. Given IID settings, submitted models can also be compared to each other [5], [6], [18] or with a reference model [19], [20], using distance metrics. Among these, `FLAME` [18] stands out, as it leverages multiple complementary methods to stop malicious participants: clustering to identify *multiple* groups of attackers, norm-clipping to mitigate gradient boosting attacks, and adaptive noising to lessen the impact of outliers. Yet, because it works under the assumption that the biggest cluster represents benign participants and that attackers cannot exceed 50% of the population, `FLAME` *de facto* falters against a majority of malicious clients. Furthermore, while the paper demonstrates that it can resist to low proportions of *non-IID* participants, it still aims at delivering one common global model, thus failing to address the more skewed non-IID cases, where leveraging multiple sub-federations might be necessary.

The assumption of IID data rarely holds in FL, even though its properties facilitate the detection of Byzantine participants. Indeed, given non-IID settings, You *et al.* [21] show most of these mitigation strategies are inefficient. These methods rely on a single source of truth that may be known beforehand [6], or elected among participants [5]. However, by definition, this single source of truth does not exist in non-IID datasets. To circumvent this issue, `FoolsGold` [7] and `CONTRA` [8] assume that sybils share a common goal, and thus produce similar model updates, allowing to distinguish them from benign non-IID participants that present dissimilar contributions. Similar participants are classified as sybils using the cosine similarity between gradient updates, and their weight is reduced in the final aggregation. However, while this mitigation strategy works when multiple attackers collaborate, it fails at identifying lone attackers. These approaches are also well suited for *pathological non-IID* scenarios, where all participants are significantly different. In *practical non-IID* settings, legitimate communities of similar participants can exist. Those legitimate participants would be falsely identified as sybils.

Finally, Zhao *et al.* [22] take a different approach and rely on client-side evaluation. Local models are aggregated into multiple sub models, which are then randomly attributed to multiple clients for efficiency validation. To also address non-IID datasets, clients self-report the labels on which they have enough data to conduct an evaluation. While this self-reporting limits the network and client resources consumption, abusive self-reporting is possible. Nevertheless, directly leveraging the participant datasets for evaluation removes the need for a single exhaustive source of truth. Resource consumption is also less of an issue in cross-silo use cases: they often imply fewer participants, with more data and dedicated resources.

B. Clustered Federated Learning

Non-IID data can also be regarded as heterogeneous data distribution that are regrouped together. Following this idea, some works [23]–[26] try to group participants sharing similarities. The purpose of this approach is twofold. First, from a performance perspective, outliers that do not fit in any group slow down the convergence [26]. Second, considering outliers as poisoned models [23] allows poisoning detection. Since the effective number of clusters is unknown, hierarchical clustering is a common way to create appropriate clusters [24], [26]. Specifically, Ye *et al.* [26] use the cosine similarity of local models to successfully group participants in more homogeneous subgroups. However, as this approach doesn't aim to address Byzantines, it does not consider that some malicious participants might aim to be grouped with benign ones to poison the cluster's model.

C. Reputation Systems for Federated Learning

Reputation systems subjectively assess participants' ability to perform a task based on past interactions. FL leverages reputation systems in three different ways. Some approaches [8], [27], [28] rely on reputation to select reliable clients for the next round. `CONTRA` [8] works this way. By progressively penalizing the participants that propose models similar to each others, and that are thus suspected of being sybils, it leaves room for participants issuing dissimilar models to be selected more often. We detail in Section III-A the limits of these types of approaches in practical non-IID settings.

Others leverage reputation to weight local models during the aggregation process [29], [30]: the higher the reputation, the heavier the local model contributes to the aggregated model. Some will even go so far as to discard contributions when the author's reputation is too low. Finally, as shown by Karimireddy *et al.* [31], small malicious incremental changes can be small enough to be undetected in a single round but still eventually add up enough to poison the global model over the course of multiple rounds. Reputation system's ability to track clients' contributions over time [27], [30] can be used as a countermeasure to these attacks.

IV. Architecture

This section details `RADAR`'s architecture. It is divided into three main components: (*i*) our cross-evaluation scheme that provides local feedbacks on each participant's contributions (Section IV-A), (*ii*) a similarity-based clustering algorithm that groups participants based on evaluations (Section IV-B), and (*iii*) a reputation system that assesses participants' trustworthiness based on their past contributions (Section IV-C). Figure 1 depicts the overview of `RADAR`.

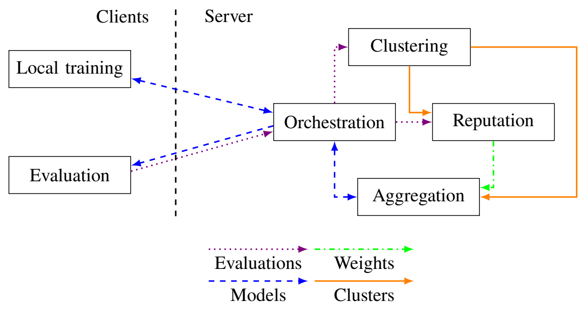


Fig. 1: *Architecture overview.*

A. Assessing Contributions with Cross-Evaluation

As highlighted in Section III, most related works on poisoning mitigation in FL rely on server-side models comparison [7], [8]. They measure distance between the parameters (for DNNs, n-dimensional arrays containing the weights and biases of each neuron) using metrics such as cosine similarity [7] or Euclidean distance [32]. However, models that are statistically further from others are not automatically of poor quality. To cope with this limitation, as well as the absence of source of truth, we propose to rely on client-side evaluation [22]. The results of this evaluation can then be used by the server to either discard or weight contributions. `RADAR`'s workflow thus differs from typical approaches by adding an intermediate step for evaluating parameters:

1. *client fitting* – The server sends clients training instructions and initial parameters, *i.e.* randoms values for the first round. For subsequent rounds, the initial parameters of each client are set to the model $\overline{w}_k^{r-1}$ of the corresponding cluster, using the results of Step 3. at round $r-1$. Each client trains its own model using the provided hyperparameters, and the initial parameters as a starting point before uploading their parameters w_i^r to the server.
2. *cross-evaluation* – The server serializes all client parameters in a single list that is sent to every client. Each client then locally evaluates each received model using its validation set, generating a predefined set of metrics such as loss, accuracy, or F1-score. The metrics of all clients are then gathered server-side.
3. *parameter aggregation* – The server partitions clients into a set of clusters $\mathscr{C}^r$ based on the evaluations gathered in Step 2. For each cluster $C_k^r \in \mathscr{C}^r$, the server computes the new model $\overline{w}_k^r = \sum_{i \in C_k^r} w_i^r \rho_i^r$, where the weight ρ_i^r is given by the reputation system for the participant p_i.

The cross-evaluation step generates an evaluation matrix that is used twice in the architecture. Since this matrix is not symmetric, the vector of *issued evaluations* $E_{[i,*]}^r$ is used for clustering, while both the *received evaluations* vector $E_{[*,j]}^r$ and the *issued evaluations* vector $E_{[i,*]}^r$ are used in the reputation system. Algorithm 1 (in Appendix) details the proposed workflow.

B. Fighting Heterogeneity with Clustering

The clustering algorithm seeks to gather similar participants together in more homogeneous sub-federations when appropriate. Nguyen *et al.* [18] and Ye *et al.* [26] both measure participants' similarity by comparing the distance between model updates. This is biased, as models that are statically different might still produce relevant results. `RADAR` addresses this issue by defining similarity as the distance between participants emitted evaluations. Indeed, since all participants evaluate the same models, the variation in evaluation results reflects a difference in the evaluation datasets. Therefore, participants having similar datasets should issue similar evaluations.

We note $\delta^r_{i,j}$ the distance between the evaluations of p_i and p_j at round r. $\delta^r_{i,j}$ is defined as the cosine similarity between p_i and p_j issued evaluation vectors $E^r_{[i,*]}$ and $E^r_{[j,*]}$, or $\delta(E^r_{[i,*]}, E^r_{[j,*]})$. We then iteratively group similar participants into different clusters, leveraging hierarchical clustering. Initially, each participant is assigned to a different cluster. Then, each closest pair of clusters is merged, thus reducing the number of clusters. The process is repeated until the distance between the two closest clusters exceeds a given threshold.

While hierarchical clustering does not require the number of clusters as an input, choosing the right threshold can be challenging. Contrarily to Ye *et al.* [26] who manually adjust this parameter on a per-dataset basis, `RADAR` leverages a dynamic threshold based on the mean inter-distance $\overline{\Delta^r}$ between the clusters at round r. This threshold θ is expressed as:

$$\theta = \beta\overline{\Delta^r} = \frac{\beta}{|\mathscr{C}^r|(|\mathscr{C}^r|-1)} \sum_{\substack{k,\ell\in\mathscr{C}^r,\\ k\neq l}} \Delta^r_{k,\ell} \tag{2}$$

where β is a tunable hyperparameter, and $\Delta^r_{k,\ell}$ the distance between two clusters C^r_k and C^r_ℓ, defined as the distance between their centroids: $\delta(\mu^r_k, \mu^r_\ell)$. The centroid μ^r_k of a cluster C^r_k is the average of the issued evaluations from its participants at round r, *i.e.*, we have $\mu^r_k = \frac{1}{|C^r_k|}\sum_{i\in C^r_k} E^r_{[i,*]}$.

Based on the results of the clustering, the server can then aggregate the models of each cluster C^r_k separately, using the reputation system described in Section IV-C. Consequently, the server maintains as many global models $\overline{w}^r_k$ as there are clusters at each round. Note that this is another difference with `FLAME` [18], which only produces a single common model for every participant.

C. Ensuring Quality Contributions with Reputation

The reputation system centrally computes the weights $\rho^r_i, \forall p_i \in C^r_k$ used in the aggregation of each cluster model $\overline{w}^r_k$ at round r (see Section IV-A). Given the existence of methods for common tasks, such as contribution filtering, `RADAR` models trust using a multivalued Dirichlet probability distribution [33]. However, the evaluations $E^r_{[*,i]}$ received by a participant p_i are continuous over $[0,1]$, and thus need to be discretized into a set of q possible values $\mathcal{E} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_q\}$.

A Dirichlet distribution on the outcome of an unknown event (*i.e.*, the mean of the received evaluation $\frac{1}{n}\sum_{e^r_{i,j}\in E^r_{[*,j]}} e^r_{i,j}$) is usually based on the combination of an initial belief vector and a series of cumulative observations [33]. As a complete cross evaluation is already available at the first round, `RADAR` does not require an initial belief vector to bootstrap reputation.

Following the notation used by Fung, Zhang, *et al.* [33], we note $\vec{\gamma}^r = \{\gamma^r_1, \gamma^r_2, \ldots, \gamma^r_q\}$ the cumulative evaluations received by p_i: $\gamma^r_2 = 3$ means that three evaluations in $E^r_{[*,j]}$ had values bounded by $\left[\frac{1}{q}, \frac{2}{q}\right[$. We then note $\vec{\mathbb{P}} = \langle\mathbb{P}\{\varepsilon_1\}, \mathbb{P}\{\varepsilon_2\}, \ldots, \mathbb{P}\{\varepsilon_q\}\rangle$ the probability distribution vector for the received evaluation of a participant, where $\sum_{s=1}^{q}\mathbb{P}\{\varepsilon_s\} = 1$. Leveraging the cumulative evaluations $\vec{\gamma}^r$, the probability $\mathbb{P}\{\varepsilon_s|\vec{\gamma}^r\}$ is given by $\mathbb{P}\{\varepsilon_s|\vec{\gamma}^r\} = \gamma_s/\sum_{m=1}^{q}\gamma_m$.

The system further needs to limit the ability of potential malicious participants to manipulate their evaluations, either by badmouthing another participant, or by artificially raising their own ratings. Consequently, the evaluations issued by a participant $p_i \in C^r_k$ are weighted according to their similarity with other cluster members' [34] as $e'_{i,j} = e^r_{i,j} sim(E^r_{[i,*]}, E^r_{[C^r_k,*]})$, where the similarity is defined as:

$$sim(E^r_{[i,*]}, E^r_{[C^r_k,*]}) = 1 - \sqrt{\frac{\sum_{j=1}^{n}\left(e^r_{i,j} - \sum_{i\in C^r_k}\frac{e^r_{i,j}}{|C^r_k|}\right)^2}{|P|}}. \tag{3}$$

To prevent attacks phased over multiple rounds, while preventing past mistakes from permanently impacting a participant, we use an exponential decay as forgetting factor, noted $\lambda \in [0,1]$. The reputation ψ^r_i of a participant p_i at round r based on the prior knowledge γ^r_i of this participant is given by Equation (4). Note that a small λ gives more importance to recent evaluations: $\lambda = 0$ only considers the last round while $\lambda = 1$, considers all round with equal weight. Based on ψ^r_i, the weight ρ^r_i of w^r_i for aggregation in $\overline{w}^r_k$ (see Step 3. in Section IV-A) is given by Equation (5).

$$\psi^r_i = \sum_{\kappa=1}^{r}\lambda^{r-\kappa}\gamma^\kappa_i \tag{4}$$

$$\rho^r_i = \frac{\psi^r_i}{\sum_j^{|C^r_k|}\psi^r_j} \tag{5}$$

As such, the weight ρ^r_i of p_i will be proportional to its reputation, and therefore the evaluations it received over time. The attackers' evaluations only vary on the subset of samples that are impacted. Consequently, the differences between their reputation scores and those of legitimate participants can be relatively small, despite remaining meaningful. We apply a sigmoid function to convert these scores to aggregation weighs and accentuate this difference. This sigmoid function is the normal distribution cumulative density function adjusted with the σ parameter.

V. Experimental Setup

We evaluate `RADAR` and any selected baseline on a set of heterogeneous intrusion detection datasets [35] with various attack scenarios (see Section V-B). We implement the described use case (Section II-A) and threat model (Section II-C) as a

TABLE I: *Hyperparameters*. The model's configuration is taken from the work of Popoola *et al.* [37], while the parameters for RADAR's architecture have been selected empirically.

Model hyperparameters	
Learning rate	0.0001
Batch size	512
Hidden layers activation	ReLU
Output layer activation	Sigmoid
# Input features	49
# Hidden layers	2
# Neurons (hidden layers)	128
Optimization algorithm	Adam
Loss function	Log loss
Number of local epochs	10

Clustering hyperparameters	
Distance metric	Cosine similarity
Threshold factor β	0.25
Cross-eval metric	F1-score

Reputation hyperparameters	
Number of classes	10000
History parameter λ	0.3
Cross-eval metric	F1-score
Normal distribution σ	0.0005

TABLE II: *Cross evaluation (F1-score) on the used datasets.* Each dataset is uniformly partitioned into a training set (80%) and an evaluation set (20%). The same partitions are kept over the entire experiment. Each model (rows) is trained on its training set during 10 epochs, and then evaluated on each test set (columns). The highest scores are highlighted in bold.

Training set	Evaluation set: CIC-IDS	NB15	ToN_IoT	Bot-IoT
CIC-IDS	**0.961787**	0.002723	0.524219	0.680166
NB15	0.108913	**0.947204**	0.009875	0.655943
ToN_IoT	0.211792	0.419380	**0.966679**	0.081510
Bot-IoT	0.158477	0.017188	0.703195	**0.999483**

set of experiments using the FL framework Flower [36], with Nix and Poetry to reproducibly manage dependencies. The hyperparameters used in our setup are detailed in Table I. The code for all experiments can be found online[1], with configuration and seeds for each considered baseline and evaluation scenario. We also provide lock files to enable anyone to reuse the same software versions as in this paper.

A. Datasets and local algorithm

To create groups of participants that share similar distributions, we use the standard feature set for flow-based NIDSs proposed by Sarhan *et al.* [35], which is based on the NetFlow v9 format from nProbe [38]. The authors converted four known IDS datasets to this format: UNSW-NB15 [39], Bot-IoT [40], ToN_IoT [41], and CSE-CIC-IDS2018 [42]. The uniform feature set allows evaluating FL approaches on independently generated datasets [37], [43]. Each contains benign samples and multiple attack classes. For instance, Bot-IoT is divided as "Benign", "DoS", "DDos", "Reconnaissance", and "Theft".

We use the "sampled" version (1,000,000 samples per dataset) provided by the same team [44]. Like Carvalho Bertoli *et al.* [43], we remove source and destination IPs and ports, as they are more representative of the testbed environment than of the traffic behavior. We then use one-hot encoding[2] on the categorical features (both for samples and labels), and apply min-max normalization to give all features the same importance in model training.

[1]Link available upon publication.

[2]Binary representation of categorical variables used in ML where each unique category is represented in binary by a zeroed vector with a *one* for the corresponding category.

Locally, we use a multilayer perceptron (MLP) with two hidden layers, following Popoola *et al.* [37]. We reuse the hyperparameters provided by the authors (see Table I), and reproduce their results on our implementation, using the same four datasets. Their algorithm shows low performance when training the model on one dataset, and evaluating it on another, as illustrated in Table II. This supports the assumptions behind the cross-evaluation proposal, where the differences between the evaluation results can be used to estimate the similarity between the local data distribution.

B. Evaluation scenarios

The threat model defined in Section II-D is implemented as a set of evaluation scenarios which model various data-quality situations. These scenarios can be summarized in three categories:

C1: `Benign.` This category actually contains one scenario which showcases a *practical non-IID* situation, where participants can be grouped into 4 use cases. Each of the 4 datasets described in V-A is randomly distributed among 5 participants without overlap. We thus have a total of 20 participants with different data, but some share similarities between their data distributions.

C2: `Lone Byzantine.` The scenarios in this category differ from the `Benign` category (C1) by introducing a fault in a single participant. This fault might be due to an *honest-but-neglectful* participant that misclassified samples or missed an intrusion, or a single *malicious participant* actively trying to poison the system. We emulate the fault by flipping the one-hot encoded label on a subset of the participant's data: given a label $\vec{y} \in \{\langle 0, 1\rangle, \langle 1, 0\rangle\}$, a faulty sample will be assigned to $\langle \neg\vec{y}_0, \neg\vec{y}_1\rangle$. A fault is characterized by two parameters:

(1) its *target*, *i.e.*, the classes to which the affected samples belong; and
(2) its *noisiness*, *i.e.*, the percentage (ranging from 10% to 100%) of targeted labels that are actually flipped.

If a single class is affected, the fault is *targeted*, and only the samples of this class see their label changed. We arbitrarily chose Bot-IoT and its "Reconnaissance" class as the target for the experiments. Otherwise, the fault is *untargeted*, and all classes of Bot-IoT are equally affected, including benign samples.

C3: `Colluding Byzantines.` This category encompasses scenarios resembling the `Lone Byzantine` ones (Category C2), but where the same fault is replicated on multiple participants at the same time. This corresponds to *malicious participants* in our threat model, as it is unlikely that several *honest-but-neglectful* participants commit the very same fault. The colluding attackers are a *majority* if they outnumber the benign participants whose data originate from the same dataset, and a *minority* otherwise.

As we experiment attacks on the Bot-IoT dataset whose data is distributed among 5 participants, this respectively means that there are three attackers and two benigns, or two attackers and three benigns. These two sub categories are referred to as `Colluding majority` and `Colluding minority`, respectively.

We note the parameters of a fault as `<noisiness><initial_of_target>`, and use this notation to refer to scenarios hereafter. As such, a `Lone 80T` scenario means that one of the five participants coming from the Bot-IoT dataset will flip 80% of its "Reconnaissance" labels to the opposite value. `Colluding minority` $\leq$`30U` refers to all scenarios where two participants from Bot-IoT flip the labels on 30% of their entire dataset, or less.

C. Metrics

To measure the ability of `RADAR` to cluster clients correctly, we use the Rand Index. The Rand index compares two partitions by quantifying how the different element pairs are grouped in each. It is defined between 0 and 1.0, 1.0 meaning that both partitions are identical. `RADAR` already produces evaluation metrics at each round thanks to the cross-evaluation scheme, based on each participant's validation set. The same evaluation methods are thus used on a common testing set (to each initial client dataset) and aggregated to evaluate the approach. The presented results focus on the mean accuracy and miss rate of the benign participants. Finally, the attack success rate (ASR) is computed over the benign participants of the affected cluster, and defined as the mean miss rate on the targeted classes of targeted attacks, and the mean of the misclassification rates (*i.e.* $1 - accuracy$) in untargeted ones.

VI. Experimental Results

`RADAR` serves multiple objectives at once: (a) maintaining high performance on *practical* non-IID data, (b) correctly identifying and weighting low-quality contributions, and (c) mitigating the impact of label-flipping attacks. As a result, we select relevant baselines from the literature to evaluate each of `RADAR`'s abilities. We use `FedAvg` [2] (abbreviated `FA`) to highlight the existing issues with statistical heterogeneity, using the setup provided by Flower [45]. Because `RADAR` can be partially assimilated as a clustered `FedAvg` variant, we also consider a theoretical setup where participants are clustered based on their original data distribution, and one instance of `FedAvg` is executed per cluster. We refer to it as *Clustered* `FedAvg` or `FC`. To highlight `RADAR`'s ability to compare with Sybil-focused mitigation strategies, we compare it with `FoolsGold` [7] (also designated `FG`). We reuse the authors' code [46], and adapt it to model updates, since `FoolsGold` was originally implemented on `FedSGD`. The following sections cover these topics using the scenarios laid out in Section V-B. Like the others, `RADAR` is abbreviated as `RA` when needed.

A. Heterogeneity

Because our use case implies that some participants share similar data distributions, we expect `RADAR`'s clustering component to limit the impact of heterogeneity by grouping similar participants together. To evaluate our approach, we compare the partition created by `RADAR`'s clustering algorithm with one where participants are grouped according to their dataset of origin. This partition is presented as Partition A in Table III. The constant Rand Index of 1.0 indicates that all participants are correctly grouped, regardless of the considered evaluation scenario. This validates the idea that similarity between evaluations can be used to regroup participants.

In addition to managing heterogeneity, it is critical that the countermeasures deployed in `RADAR` do not negatively impact performance. Specifically, the reputation system must not unfairly penalize legitimate participants because of their potential differences. Figure 2a presents the weights provided by the reputation system for the aggregation. In the `Benign` scenario, the 5 participants originating from the Bot-IoT dataset do have equal weights, confirming that none of them is penalized by the reputation system. Furthermore, Table IV indicates that `RADAR`'s mean accuracy is superior to `FoolsGold`'s and `FedAvg`, as both baselines falter in practical non-IID use cases. `RADAR` almost matches the results of `FC`, which is ideally clustered by design. Overall, `FoolsGold`, a reference Byzantine-resilient FL strategy tailored for non-IID settings, falters in *practical* non-IID settings, where `RADAR` strives.

B. Handling data quality

Another goal for `RADAR` is to handle contributions of various quality. This objective is mostly represented by scenarios of Category C2 (`Lone`), as we consider that coordinated faults are improbable for legitimate participants. In this configuration, we expect the Byzantine participant to be either, put in a cluster of its own, or penalized by the reputation system. To verify the former, we compare the partition made by `RADAR` with another where Byzantines are segregated in an additional cluster (see Partition B in Table III). Here, a Rand Index lower than 1.0 implies that Byzantine participants have been grouped with legitimate ones of the same dataset, which is the case in most scenarios of the `Lone` category. However, the noisiest untargeted faults (`Lone >95U`) result in the Byzantine participant being placed in his own separate cluster, thus neutralizing its impact on the other participants. Note that the hyperparameters of the clustering algorithm could be tuned so that attackers with lower *noisiness* would be separated, notably the threshold factor β and the cross-evaluation metric (see Table I).

When Byzantine participants are grouped with benign ones, we rely on the reputation system to identify and diminish the impact of their contributions. The weights given by the reputation system can be seen in Figure 2b, where the Byzantine client is heavily penalized in the `Lone 100T` scenario. The effect of the clustering and reputation system are also apparent in Table IV, where the ASR for both `Lone 100T` and `Lone 100U` are comparable to the benign case, underlining `RADAR` resilience. The results in Figure 3 confirm this trend: `RADAR` maintains a low ASR in most configurations. As a result, `RADAR` demonstrates its ability to mitigate isolated Byzantine faults, regardless of their intensity.

TABLE III: *Rand Index between `RADAR`'s clustering and two partitions of reference, under various scenarios.* Partition **(A)** contains only benign participants grouped according to their respective dataset. Partition **(B)** contains attackers placed in a separated group in addition to benign participants.

Scenario *Category*	*Noisiness*	*Target*	Partition (A)	Partition (B)
`Benign`			1.00	1.00
`Lone`	≤100	T	1.00	0.97
`Lone`	≤95	U	1.00	0.97
`Lone`	100	U	1.00	1.00
`Collud. min.`	≤100	T	1.00	0.97
`Collud. min.`	≤90	U	1.00	0.97
`Collud. min.`	100	U	1.00	1.00
`Collud. maj.`	≤100	T	1.00	0.96
`Collud. maj.`	≤90	U	1.00	0.96
`Collud. maj.`	100	U	1.00	1.00

TABLE IV: *Effect of different attack configurations (`100T/U`) on all baselines.* The attack success rate (ASR) is computed over the targeted classes in targeted attacks, and over all samples otherwise (see Section V-C). `RA` is `RADAR`, `FG` is `FoolsGold`, `FA` is `FedAvg` (on *all* participants), and `FC` is `FedAvg` ideally clustered per dataset. The ASR of benign runs is provided as a baseline. `RADAR`'s limiting scenario is marked ‡.

Scenario	Mean accuracy (%) RA	FG	FA_a	FC	ASR (%) RA	FG	FA	FC
Targeted (`100T`)								
`Benign`	99.07	55.04	79.49	**99.24**	**0.00**	5.17	5.10	0.09
`Lone`	99.06	60.51	77.38	**99.22**	**0.00**	93.82	6.73	0.45
`Collud. min.`	**98.96**	54.64	78.48	98.33	**0.00**	2.97	9.99	53.40
‡ `Collud. maj.`	**98.28**	85.10	79.40	98.22	73.39	**8.10**	17.65	59.36
Untargeted (`100U`)								
`Benign`	99.07	55.04	79.49	**99.24**	0.09	0.39	33.30	**0.06**
`Lone`	98.96	49.56	78.38	**99.22**	**0.08**	99.89	54.70	0.12
`Collud. min.`	**98.98**	49.67	72.47	97.69	0.10	**0.04**	44.53	6.26
`Collud. maj.`	**98.96**	69.09	81.87	75.66	**0.08**	38.98	59.49	94.36

The same cannot be said for `FoolsGold`'s, which aims at providing a single global model. Further, by construction, it identifies groups of similar participants as colluding attackers and considers that only the faulty participant is legitimate. This appreciation error leads `FoolsGold` to have the worst attack success rate among all tested baselines, even when compared against the naive `FedAvg` approach.

C. Label flipping attacks

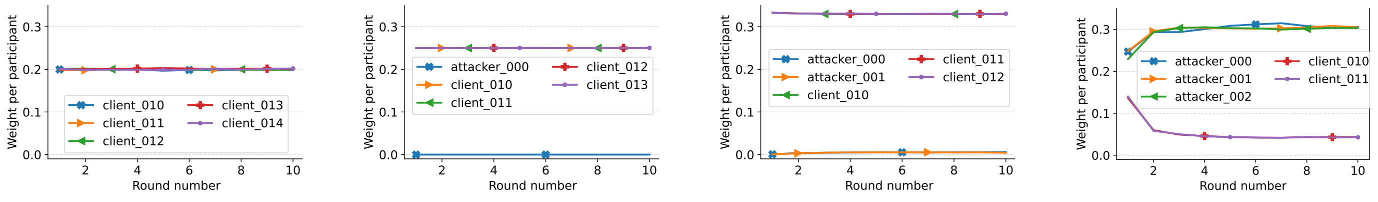
We evaluate the resistance to label-flipping attacks using two different scenarios. First, we consider that `Colluding Byzantines` can only refer to attackers, as it is unlikely that the very same fault happens over multiple clients at the same time. Second, the `Lone 100U` scenario, as it is similarly unlikely that for an *honest-but-neglectful* participant to misclassify the entirety of its data.

Like discussed in Section VI-B, the clustering algorithm separates the noisiest attacks from the rest. This is true regardless of the number of attackers, as confirmed by the results in Table III. For untargeted faults with at least 95% *noisiness*, the Rand Index at round 10 stays equal to 1.0. This means that for those loud attacks, attackers are separated from benign participants, hence negating their poisoning effect. This is a critical result for `RADAR`, as this mitigation occurs for *any number of attackers*, even if they outnumber benign participants. However, the attackers in `Colluding T` scenarios are placed with legitimate participants in the same cluster.

Minority of attackers: The `Colluding minority` class (Category C3) contains scenarios where 2 out of 5 participants instantiated in Bot-IoT perpetrate label-flipping attacks. Here, the results depicted in Figure 2c indicate that the attackers are heavily penalized by the reputation system. This is coherent with the results in Table IV for these scenarios, where we can see that `RADAR` indeed fend off attackers with an ASR of 0.0. Among the other baselines, `FedAvg` is especially affected, since it does not have any protection against such attacks. This is also true for our theoretical baseline `FC`, although the effect is logically limited to participants using the Bot-IoT dataset. `FoolsGold`, on the other hand, detects the attackers since they are similar and thus manages to discard the attack, obtaining a rather low ASR of 2.97%. Unfortunately, it also detects benign members from the other clusters as colluding attackers and thus train on BoT-IoT only, leading to a very low 54.64% accuracy overall.

Majority of attackers: The `Colluding majority 100T` scenario, with 3 attackers out of 5 participants, sees the attackers gain precedence. Figure 2d clearly illustrates this phenomenon, where the legitimate participants' weights drop as the reputation system favors the attackers. This is a known limit of the reputation system, which favors the majority by construction. This is further illustrated in Figure 5: a steeper drop in accuracy and miss rate occurs when attackers outnumber benign participants in one cluster. However, the metric distribution over the participants highlights that the other clusters remain unaffected, and that the majority of benign participants continues to perform well. Furthermore, as illustrated in Figures 3 and 4, the *noisiness* of attackers must exceed 80% for attackers to poison the cluster's model. Consequently, while this scenario highlights a limitation of `RADAR`, it is significantly constrained.

Impact of the attack timing: Additionally, Figure 6 depicts how the reputation system reacts to participants that change their *noisiness* over time. Figure 6a features a `Colluding minority 100T` scenario where the noisiness drops to 0% at round 3. The system forgives attackers approximately four rounds after they adapted their behavior. This rather short delay depends on the chosen λ history parameter of our reputation system (see Table I). On the contrary, Figure 6b showcases `Colluding minority T` attackers going from 0 to 100% noisiness over the course of a few rounds. The reputation system detects and penalizes them at round 5 when the noisiness reaches 60%. This in phase with the conclusions of Figures 3



(a) Benign. (b) Lone 100T. (c) Colluding minority 100T. (d) Colluding majority 100T.

Fig. 2: *Aggregation weights ρ_i^r for the participants coming from the BoT-IoT dataset depending on the number of Byzantines (*`100T`*).* Byzantines are correctly penalized when they are a minority, but gain precedence when they become the majority.

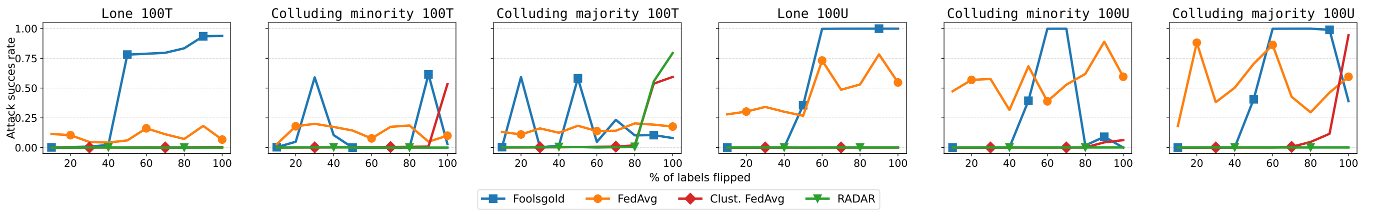


Fig. 3: *Attack success rate (ASR) of the different baselines.* Even though attackers are a majority, they gain weight precedence only for higher poisoning rates (>90%).

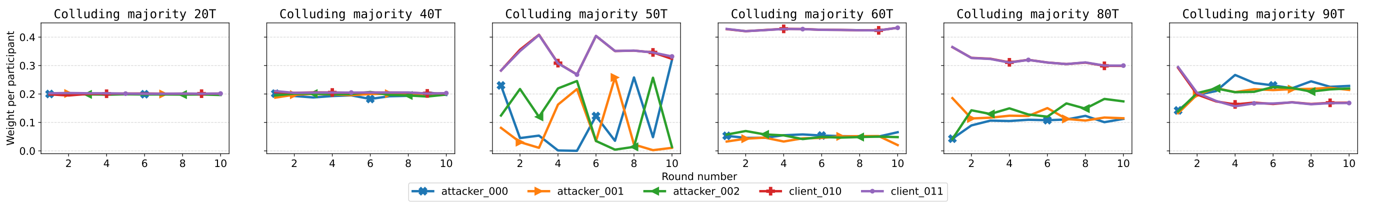


Fig. 4: *Aggregation weights ρ_i^r per participant of the poisoned cluster (*`Colluding majority T`*).* Even though attackers are a majority, they gain weight precedence only for higher poisoning rates (≥90%).

and 4: for lower noisiness levels, the attackers have no effect. The reputation system thus detects attackers only when they start to present a threat to the global model's performance.

D. Synthesis

First, the results highlight the relevance of clustering in *practical non-IID* use cases, as attacks are confined to the cluster attackers have been assigned to. This is particularly visible in the performance of `RADAR` and the clustered `FedAvg` variant, which both maintain high accuracy overall by providing each community with a specific model. This is true even in the presence of Byzantine faults or attackers. However, since `FC` does not implement any mitigation strategy, its performance quickly degrades with the quality of the contributions, especially in the presence of colluding attackers (as illustrated by Figure 3).

The results in Table IV also emphasize on `FoolsGold`'s unsuitability for *practical non-IID* use cases, where groups of participants sharing similar distributions can exist. Especially in a `Lone` scenario, any groups of similar participants are considered as colluding attackers and penalized, leading to high ASR, as only the attacker is considered as legitimate. Similarly, in `Colluding majority T/U` scenarios, `FoolsGold` penalizes all the other clusters, leading to a model trained on Bot-IoT only. Overall, `RADAR` presents the most consistent results, with high accuracy and low ASR in most scenarios, only failing against a majority of extremely *noisy* colluding attackers that still managed to get similar enough to be grouped with benign participants.

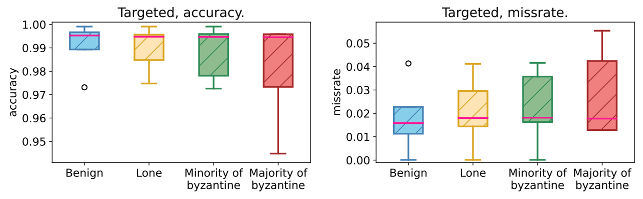
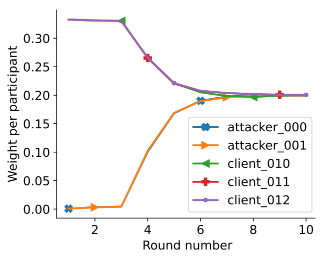
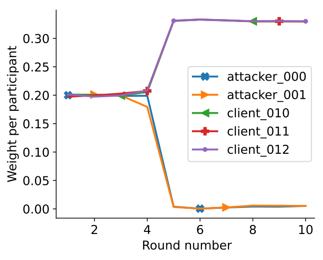


Fig. 5: *`RADAR`'s metric distribution among participants in different scenarios (*`100T`*).* The accuracy's and miss rate's lower bounds suddenly drop when attackers outnumber benign participants in the affected cluster. Indeed, clients in other clusters are unaffected by the poisoning.



(a) Attackers act with 100% *noisiness*, but become benign on round 3.



(b) Attackers start benign, and increase *noisiness* by 20% each round when $r \geq 3$.

Fig. 6: *Aggregation weights ρ_i^r per participant of the poisoned cluster (`Colluding minority T`).* Attackers are forgiven over time, and the reputation system reacts quickly to newly detected attackers.

VII. Discussion

The experiments illustrate how `RADAR` succeeds at identifying attackers in heterogeneous context, thus demonstrating its versatility. In this section, we discuss the limitations and potential consequences of our architecture and propose research directions to close these gaps.

A. Generalizability

While the experiments are only conducted on intrusion detection datasets, `RADAR`'s design could be used in different use cases regarding the following conditions: (1) parametric local models whose parameters can be aggregated using FL, and (2) local testing sets and relevant metrics allowing participants to evaluate the others' models. Since the NIDS use case induces a focus on malicious samples (*i.e. positive* values), we choose the F1-score as input for our clustering and reputation algorithms, as it emphasizes on false positives and false negatives. However, `RADAR` can handle different metrics, for instance the loss of a model during evaluation, particularly relevant for similarity measurements.

B. Scalability and performance

The focus on small-scale collaboration (*i.e.* a few dozens of participants) makes the overhead of the *cross-evaluation* step (Section IV-A) practical, and justifies the absence of performance-related metrics in this paper. However, one can question the scalability of the proposed approach in larger scale applications. Indeed, at each round, clients evaluate $|P|$ additional models, which scales linearly with the number of clients. Two new communications are also introduced, one to send the models and one to collect the evaluations. Their size also grows linearly with $|P|$, as the models of all participants must be evaluated. Likewise, we exclude execution-related performance evaluation such as training time, CPU overhead, or bandwidth consumption. It opens the way to interesting research directions on how to implement and scale `RADAR` while guarantying its properties.

C. Evaluation poisoning

Attackers could try to poison the evaluations that they provide on other participants to abuse the system. However, the implementation presented in Section V implies that attackers poison both their training and testing sets. Consequently, the evaluations they produce on other participants are directly affected. We thus expect the system to cope with arbitrary poisoning similarly to data poisoning: either by placing the attackers in a different cluster because of their dissimilarity, or by penalizing their reputation.

D. Information disclosure

Because `RADAR` shares models with the other participants to obtain feedbacks, it can be argued that it revels more information about the participants. This is limited to the participants' models, which are shared without identifiers. However, since clients also receive the global model of their cluster, they can try to estimate the models that belong to their cluster. This remains challenging, as the models are weighted using the reputation score of the participants, which are only available to the server. Comparing the privacy impact of `RADAR` with those of simpler approaches like `FedAvg` represents interesting research directions.

VIII. Conclusion

In this paper, we introduced `RADAR`, a federated learning framework that effectively deals with Byzantine participants, even with heterogeneous data-distributions. Our approach is built on the assumption that heterogeneous participants can be grouped based on the similarity between their data distribution. This assumption is validated through experiments relying on four different public datasets, each dataset corresponding to a different client use case. We introduce a cross-evaluation scheme that allows participants to measure their pairwise similarities. Based on those measurements, we manage to rebuild the initial participant distribution using hierarchical clustering. Our results confirm that evaluation metrics can indeed be used to assess similarity between participants, without accessing their datasets nor comparing their models statistically.

We further designed a reputation system based on the cross-evaluation results. Our reputation system uses the perceived similarity of participants and their cumulated past results to give a score to each participant inside a cluster. We are able to validate that the combination of the clustering and reputation system can mitigate all tested Byzantines scenarios, with the single exception of targeted attacks where a majority of Byzantines flip more than 80% of their labels. We compared our work to `FoolsGold` and `FedAvg`, which highlighted the versatility of `RADAR`.

Finally, the construction of the proposed architecture, with indirect feedbacks and personalized model weighting, makes it a suitable candidate for decentralized architectures. In this regard, being able to remove the central server dependency is a key step towards a truly decentralized, trustworthy, and privacy-preserving collaborative machine learning framework.

References

[1] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, Mar. 8, 2021.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, Communication-efficient learning of deep networks from decentralized data. In *20th international conference on artificial intelligence and statistics*, 2017.

[3] L. Lavaur, M.-O. Pahl, Y. Busnel, and F. Autrel, The Evolution of Federated Learning-based Intrusion Detection and Mitigation: a Survey. *IEEE Transactions on Network and Service Management. TNSM*, TNSM, Jun. 2022.

[4] B. Pejó and G. Biczók, Quality Inference in Federated Learning With Secure Aggregation. *IEEE Transactions on Big Data*, vol. 9, no. 5, Oct. 2023.

[5] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, Machine learning with adversaries: Byzantine tolerant gradient descent. *Advances in Neural Information Processing Systems*, 2017.

[6] X. Cao, M. Fang, J. Liu, and N. Z. Gong, FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. In *28th Annual Network and Distributed System Security Symposium*, ser. NDSS, 2022.

[7] C. Fung, C. J. M. Yoon, and I. Beschastnikh, The limitations of federated learning in sybil settings. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID, 2020.

[8] S. Awan, B. Luo, and F. Li, CONTRA: Defending Against Poisoning Attacks in Federated Learning. In *26th European Symposium on Research in Computer Security*, ser. ESORICS, 2021.

[9] C. V. Zhou, C. Leckie, and S. Karunasekera, A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security*, no. 1, Feb. 2010.

[10] Y. Huang, L. Chu, Z. Zhou, L. Wang, J. Liu, J. Pei, and Y. Zhang, Personalized Cross-Silo Federated Learning on Non-IID Data. *Proceedings of the AAAI Conference on Artificial Intelligence. AAAI*, May 18, 2021.

[11] A. Jain, H. Patel, L. Nagalapatti, N. Gupta, S. Mehta, S. Guttula, S. Mujumdar, S. Afzal, R. Sharma Mittal, and V. Munigala, Overview and Importance of Data Quality for Machine Learning Tasks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20, Aug. 20, 2020.

[12] Y. Deng, F. Lyu, J. Ren, Y.-C. Chen, P. Yang, Y. Zhou, and Y. Zhang, FAIR: Quality-Aware Federated Learning with Precise User Incentive and Model Aggregation. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, May 10, 2021.

[13] Y. Deng, F. Lyu, J. Ren, H. Wu, Y. Zhou, Y. Zhang, and X. Shen, AUCTION: Automated and Quality-Aware Client Selection Framework for Efficient Federated Learning. *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, Aug. 2022.

[14] M. Fang, X. Cao, J. Jia, and N. Gong, Local Model Poisoning Attacks to Byzantine-Robust Federated Learning. In *29th USENIX Conference on Security Symposium*, ser. USENIX Security, 2020.

[15] V. Tolpegin, S. Truex, M. E. Gursoy, and L. Liu, Data Poisoning Attacks Against Federated Learning Systems. In *25th European Symposium on Research in Computer Security*, ser. ESORICS, 2020.

[16] A. N. Bhagoji, S. Chakraborty, P. Mittal, and S. Calo, Analyzing Federated Learning through an Adversarial Lens. In *36th International Conference on Machine Learning*, ser. ICML, 2019.

[17] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access*, vol. 7, 2019.

[18] T. D. Nguyen, P. Rieger, H. Chen, *et al.*, FLAME: Taming Backdoors in Federated Learning., presented at the 31st USENIX Security Symposium (USENIX Security 22), 2022.

[19] Q. Xia, Z. Tao, and Q. Li, ToFi: An Algorithm to Defend Against Byzantine Attacks in Federated Learning. In *Security and Privacy in Communication Networks*, 2021.

[20] J. Zhou, N. Wu, Y. Wang, S. Gu, Z. Cao, X. Dong, and K.-K. R. Choo, A Differentially Private Federated Learning Model against Poisoning Attacks in Edge Computing. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[21] X. You, Z. Liu, X. Yang, and X. Ding, Poisoning attack detection using client historical similarity in non-iid environments. In *2022 12th International Conference on Cloud Computing, Data Science & Engineering*, ser. Confluence, 2022.

[22] L. Zhao, S. Hu, Q. Wang, J. Jiang, C. Shen, X. Luo, and P. Hu, Shielding Collaborative Learning: Mitigating Poisoning Attacks through Client-Side Detection. *IEEE Transactions on Dependable Secure Computing*, 2020.

[23] N. Peri, N. Gupta, W. R. Huang, L. Fowl, C. Zhu, S. Feizi, T. Goldstein, and J. P. Dickerson, Deep k-NN Defense Against Clean-Label Data Poisoning Attacks. In *Computer Vision – ECCV 2020 Workshops*, 2020.

[24] C. Briggs, Z. Fan, and P. Andras, Federated learning with hierarchical clustering of local updates to improve training on non-IID data. In *International Joint Conference on Neural Networks*, ser. IJCNN, 2020.

[25] X. Ouyang, Z. Xie, J. Zhou, J. Huang, and G. Xing, ClusterFL: A similarity-aware federated learning system for human activity recognition. In *19th Annual International Conf. on Mobile Systems, Applications, and Services*, ser. MobiSys, 2021.

[26] C. Ye, H. Zheng, Z. Hu, and M. Zheng, PFedSA: Personalized Federated Multi-Task Learning via Similarity Awareness. In *IEEE International Parallel and Distributed Processing Symposium*, ser. IPDPS, 2023.

[27] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, Reliable Federated Learning for Mobile Networks. *IEEE Wireless Communications*, 2020.

[28] X. Tan, W. C. Ng, W. Y. B. Lim, Z. Xiong, D. Niyato, and H. Yu, Reputation-Aware Federated Learning Client Selection based on Stochastic Integer Programming. *IEEE Transactions on Big Data*, 2022.

[29] N. Wang, Y. Xiao, Y. Chen, Y. Hu, W. Lou, and Y. T. Hou, FLARE: Defending Federated Learning against Model Poisoning Attacks via Latent Space Representations. In *ACM on Asia Conf. on Computer and Communications Security*, 2022.

[30] Y. Wang and B. Kantarci, Reputation-enabled Federated Learning Model Aggregation in Mobile Platforms. In *IEEE International Conference on Communications*, ser. ICC, 2021.

[31] S. P. Karimireddy, L. He, and M. Jaggi, Learning from History for Byzantine Robust Optimization. In *38th International Conference on Machine Learning*, ser. ICML, 2021.

[32] Z. Ma, J. Ma, Y. Miao, Y. Li, and R. H. Deng, ShieldFL: Mitigating Model Poisoning Attacks in Privacy-Preserving Federated Learning. *IEEE Transactions on Information Forensics and Security*, 2022.

[33] C. J. Fung, J. Zhang, I. Aib, and R. Boutaba, Dirichlet-Based Trust Management for Effective Collaborative Intrusion Detection Networks. *IEEE Transactions on Network and Service Management*, 2011.

[34] L. Xiong and L. Liu, PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. *IEEE Transactions on Knowledge and Data Engineering*, 2004.

[35] M. Sarhan, S. Layeghy, and M. Portmann, Towards a Standard Feature Set for Network Intrusion Detection System Datasets. *Mobile Networks and Applications*, 2021.

[36] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, and N. D. Lane, Flower: A friendly federated learning research framework. 2020.

[37] S. I. Popoola, G. Gui, B. Adebisi, M. Hammoudeh, and H. Gacanin, Federated Deep Learning for Collaborative Intrusion Detection in Heterogeneous Networks. In *IEEE 94th Vehicular Technology Conference*, ser. VTC2021-Fall, 2021.

[38] ntop, Nprobe documentation.

[39] N. Moustafa and J. Slay, UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *2015 Military Communications and Information Systems Conference*, ser. MilCIS, Nov. 2015.

[40] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. *Future Generation Computer Systems*, 2019.

[41] N. Moustafa, M. Keshky, E. Debiez, and H. Janicke, Federated TON_IoT Windows Datasets for Evaluating AI-Based Security Applications. In *IEEE 19th International Conf. on Trust, Security and Privacy in Computing and Communications*, ser. TrustCom, 2020.

[42] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *4th International Conference on Information Systems Security and Privacy*, 2018.

[43] G. de Carvalho Bertoli, L. Alves Pereira Junior, O. Saotome, and A. L. dos Santos, Generalizing intrusion detection for heterogeneous networks: A stacked-unsupervised federated learning approach. *Computers & Security*, 2023.

[44] S. Layeghy and M. Portmann, On Generalisability of Machine Learning-based Network Intrusion Detection Systems. *Computers and Electrical Engineering*, 2022.

[45] Flower Labs GmbH., `fedavg.py` (flower). https://github.com/adap/flower/blob/main/src/py/flwr/server/strategy/fedavg.py, Jan. 5, 2024.

[46] C. Fung, C. J. M. Yoon, and I. Beschastnikh, `deep-fg/` (foolsgold). https://github.com/DistributedML/FoolsGold/tree/master/deep-fg, May 15, 2019.

Appendix

Algorithm 1 `RADAR`. R is the number of rounds, β the local batch size, η the learning rate, $\mathcal{E}$ the number of epochs, and λ a loss function.

```
Require: $P$
 1: with $r \leftarrow 0$ do
 2:     $\mathscr{C}^r \leftarrow \{P\}$
 3:     $\overline{W}^r \leftarrow (\textsc{Random}(\ ))$

 4: for $r \leftarrow 1, \dots, R$ do
 5:     ▷ Step (1): model training ◁
 6:     for all $p_i \in P$ in parallel do
 7:         $k \leftarrow \textsc{GetCluster}(p_i, \mathscr{C}^r)$
 8:         $w_i^r \leftarrow \textsc{ClientFit}(p_i, \overline{w}_k^r)$

 9:     $W^r \leftarrow (w_i^r)_{i \in n[\![1,n]\!]}$

10:     ▷ Step (2): cross-evaluation ◁
11:     for all $p_i \in P$ in parallel do
12:         $(e_{i,j}^r) \leftarrow \textsc{ClientEvaluate}(p_i, W^r)$
13:     $E_{[i,j]}^r = [e_{i,j}^r]_{i,j \in [\![1,n]\!]}$

14:     ▷ Step (3): parameters aggregation ◁
15:     $\mathscr{C}^r \leftarrow \textsc{ComputeClusters}(E^r)$          ▷ See: Section IV-B
16:     for all $C_k^r \in \mathscr{C}^r$ do
17:         $(\rho_i^r) \leftarrow \textsc{ComputeReput}(E^r, \mathscr{C}^r)$          ▷ See: Section IV-C
18:         $\overline{W}^r \leftarrow \frac{1}{|C_k^r|} \sum_{i=0}^{|} C_k^r | w_i^r$

19: function ClientFit($p$, $\omega$)          ▷ On client.
20:     for $i \leftarrow 1, \dots, \mathcal{E}$ do
21:         for all $b \in \textsc{split}(d_i, \beta)$ do
22:             $\omega \leftarrow \omega \nabla \lambda(\omega; b)$          ▷ See: Section II-A
23:
24:     return $\omega$

25: function ClientEvaluate($p$, $\Omega$)          ▷ On client.
26:     for all $\omega_j \in \Omega$ do
27:         $e_{i,j}^r \leftarrow \textsc{Eval}(\omega, d_i)$
28:     return $(e_{i,j}^r)_{i,j \in [[1,n]]}$
```