2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W) | 979-8-3315-1205-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/DSN-W65791.2025.00079

A Lightweight Reputation-Based Mechanism for Incentivizing Cooperation in Decentralized Federated Learning

Kavindu Herath*, Suraj Mahangade*, Saurabh Bagchi
{kherathm, smahanga, sbagchi}@purdue.edu
Department of Electrical and Computer Engineering, Purdue University,
West Lafayette, USA

***Abstract*—Federated Learning (FL) enables collaborative model training without centralizing data, but traditional server-assisted FL faces challenges such as single points of failure, communication bottlenecks, and trust dependency. Decentralized Federated Learning (DFL) addresses these issues by eliminating the central server, allowing peer-to-peer communication among clients. However, DFL introduces new vulnerabilities, including exposure to adversarial attacks and the risk of free-riding behavior. To ensure sustainable and secure participation, effective incentive and reputation mechanisms are essential. In this work, we propose a lightweight, reputation-based algorithm tailored for resource-constrained edge devices in decentralized settings. Our approach prioritizes model sharing based on peer-evaluated contributions, discourages selfish behavior, and mitigates the impact of malicious clients through adaptive reputation adjustments. Unlike prior methods, our scheme explicitly accounts for clients' limited sending and receiving capacities, promoting fairness and practical applicability. Experimental results demonstrate that our method incentivizes meaningful participation and enhances the robustness of decentralized learning systems against both selfish and adversarial threats.**

***Index Terms*—Decentralized Federated Learning, Reputation Mechanism, Incentive Design**

I. Introduction

Federated Learning (FL) [1] has emerged as a transformative paradigm for distributed machine learning, enabling multiple clients to collaboratively train models without sharing raw data. Originally introduced to address privacy concerns in distributed training, FL follows a centralized server-assisted framework where a central server orchestrates the learning process [1], [2]. Despite its advantages, server-assisted FL suffers from several limitations, including vulnerability to single-point-of-failure risks [3]–[5], communication bottlenecks as the number of clients scales [3], [6], [7], and trust dependency issues where clients must fully trust the server [7]–[9]. These challenges have motivated the development of alternative architectures, such as decentralized federated learning (DFL) [3], [7], [10]–[15].

DFL eliminates the need for a central server, enabling clients to communicate directly with their peers in a peer-to-peer fashion. While DFL enhances robustness and scalability, it introduces new challenges, particularly in ensuring trustworthiness and defending against adversarial behavior. Unlike server-assisted FL, where a single global model is maintained, each client in DFL not only trains locally but also acts as a parameter server, making security threats such as model poisoning more complex [16]. Consequently, robust mechanisms are required to detect and mitigate adversarial influences while preserving the decentralized nature of the system.

Prior research has explored various defenses against poisoning attacks in FL, with many Byzantine-robust algorithms proposed for server-assisted FL [17]–[22]. Beyond security concerns, the sustainability of FL and DFL hinges on incentivizing participation. Prior studies highlight the necessity of effective incentive mechanisms to encourage mobile devices to contribute computational resources for federated learning tasks [23]. Without such incentives, clients may resort to free-riding, benefiting from the trained model without actively contributing to its learning process [24]. This behavior undermines the collaborative nature of DFL and reduces the overall effectiveness of the system.

Two major types of problematic clients emerge in such settings. First, selfish clients opt not to share their model updates while still leveraging the benefits of models shared by others. Second, malicious clients pose as highly contributing participants by providing frequent model updates, but in reality, they intentionally inject adversarial modifications to degrade the model's accuracy and overall performance. These challenges necessitate an effective reputation mechanism to distinguish between cooperative and disruptive clients.

Several approaches have been proposed to address these issues. Reputation-based systems assess client reliability based on historical contributions [25], [26], yet many assume idealized conditions where all clients possess equal resources and participation capabilities [27], [28]. More recently, blockchain-based mechanisms have been explored to incentivize participation and enforce trust in decentralized systems [29]. These methods reward highly contributing clients with reputation scores or tokens while penalizing free-riders. However, blockchain operations introduce significant computa-

*These authors contributed equally as first authors.

2325-6664/25/$31.00 ©2025 IEEE
DOI 10.1109/DSN-W65791.2025.00079

tional overhead, such as consensus mechanisms and mining, which are often impractical for resource-constrained edge devices. To mitigate these challenges, many solutions offload these tasks to centralized servers, thereby reintroducing bottlenecks and partially compromising the decentralization goals of DFL.

Therefore, there is a need for a lightweight, decentralized mechanism that effectively rewards highly contributing clients while mitigating the impact of malicious participants. Such a system should ensure that trustworthy clients receive greater benefits, while preventing adversarial clients from disrupting the learning process, all without imposing excessive computational burdens on resource-limited edge devices.

To address these challenges, we introduce a simple yet robust reputation-based algorithm that offers the following key contributions:

1) We propose a straightforward reputation-based sharing scheme that efficiently operates on edge devices. This scheme ensures that clients who contribute more receive greater benefits, while selfish clients are deprioritized. By dynamically adjusting sharing priorities based on peer evaluations, our method discourages free-riding behavior and incentivizes meaningful participation.
2) To mitigate the impact of malicious clients, we introduce an adaptive reputation mechanism that dynamically adjusts based on clients' historical behavior. By leveraging local trust scores and peer evaluations, our approach effectively detects and limits the influence of both highly deviating and gradually malicious clients. This ensures that faulty or adversarial updates do not propagate unchecked, maintaining the robustness of the decentralized learning process while avoiding significant computational overhead.
3) Unlike prior studies that assume uniform resource availability, we incorporate clients' limited sending and receiving capacities into our reputation scheme. To the best of our knowledge, we are the first to model resource constraints within a decentralized reputation-based FL system, enabling more realistic and adaptive participation strategies.

II. Related Work

A. Federated Learning

Federated Learning (FL) is a distributed machine learning framework that enables multiple clients to collaboratively train a global model while keeping their local data decentralized and private. Given a set of clients $\mathcal{C} = \{C_1, C_2, \ldots, C_N\}$, each client C_i possesses its local dataset $D_i = \{(x_{i,j}, y_{i,j})\}$, where $x_{i,j}$ denotes the input data and $y_{i,j}$ represents the corresponding labels. The goal of FL is to optimize a global model θ by aggregating local model updates without sharing raw data. Let $\mathcal{L}(\theta, D_i)$ represent the loss function of client C_i for its local dataset. The local model update at each client is computed as the gradient of the local loss function:

$$\nabla \mathcal{L}_i(\theta) = \frac{1}{|D_i|} \sum_{j=1}^{|D_i|} \nabla \ell(x_{i,j}, y_{i,j}; \theta) \tag{1}$$

where $\ell(x_{i,j}, y_{i,j}; \theta)$ is the loss for the individual data point $(x_{i,j}, y_{i,j})$.

The global model θ is updated by aggregating these local updates using an aggregation rule, typically a weighted average based on the size of the local datasets:

$$\theta^{(t+1)} = \sum_{i=1}^{N} \frac{|D_i|}{\sum_{j=1}^{N} |D_j|} \theta_i^{(t)} \tag{2}$$

where $\theta_i^{(t)}$ represents the model update from client C_i at iteration t, and $\mathcal{N}$ is the total number of clients.

Despite its privacy-preserving nature, traditional server-assisted federated learning (FL) faces several limitations, including a single point of failure, communication bottlenecks, and a reliance on the central server for trust and coordination in model aggregation.

B. Decentralized Federated Learning

Decentralized Federated Learning (DFL) removes the central server and enables peer-to-peer communication among clients. In DFL, each client C_i trains its local model and participates in model aggregation with peers, updating the global model based on local exchanges. The model update rule is:

$$\theta_i^{(t+1)} = \text{Aggregate}\left(\{\theta_j^{(t)}\}_{j \in \mathcal{N}_i}\right) \tag{3}$$

where Aggregate$(\cdot)$ could be a weighted sum, median, or other methods that handle adversarial updates.

While DFL enhances scalability and robustness, it introduces new security risks, such as adversarial model poisoning, requiring robust aggregation techniques to detect and mitigate malicious behavior.

Clients in DFL also determine their participation based on available resources. Reputation-based systems can incentivize cooperation by dynamically updating each client's reputation based on past behavior, penalizing selfish or malicious clients to promote fair contributions.

C. Krum Aggregation

Krum [30] is an aggregation technique designed to mitigate the impact of adversarial updates in federated learning. The key idea behind Krum is to select the update that is most consistent with the majority of other updates, under the assumption that malicious updates will be significantly different from benign ones. Given n total clients, where c are potentially compromised, the server computes the scores of each client update Δ_i by summing the squared Euclidean distances to its $n - c - 2$ nearest neighbors. The client update with the smallest such score is selected for aggregation in the next round. Formally, Krum can be defined as:

Algorithm 1 Federated Learning Model Selection

1: **for** $t = 0$ **to** $T - 1$ **do**

2: **Step 1: Local Training**

$$x_{t+\frac{1}{2}}^{(i)} \leftarrow F(x_t^{(i)}, D_i), \quad \forall i \in \{1, \ldots, n\}$$

3: **Step 2: Requesting Models**

$$P(i,j), \quad \forall j \in \{1, \ldots, n\}, j \neq i$$

$$\text{Sorted Providers} \leftarrow \text{sort}(\{P(i,1), \ldots, P(i,n)\})_{\text{desc}}$$

$$\text{Requesting Client List} \leftarrow \text{Sorted Providers}[0 : r_i - 1]$$

4: **Step 3: Sending Models**

$$S(i,j), \quad \forall j \in \text{Requested Clients}, \quad j \neq i$$

$$\text{Sorted Clients} \leftarrow \text{sort}(\{S(i,1), \ldots, S(i,n)\})_{\text{desc}}$$

$$\text{Sending Clients List} \leftarrow \text{Sorted Clients}[0 : s_i - 1]$$

5: **Step 4: Model Aggregation**

$$\text{V}_{\text{Trimmed}} = \text{Krum}(V)_\beta \quad \forall V \in \text{Received Models}$$

$$x_{t+1}^{(i)} = \frac{a_i x_{t+\frac{1}{2}}^{(i)} + \sum_{k \in \text{Selected Clients}} a_k V_k}{a_i + \sum_{k \in \text{Selected Clients}} a_k}$$

6: **end for**

$$\text{Krum} := \left\{ \Delta_i | i = \arg\min_{i \in [c]} \sum_{i \to j} \|\Delta_i - \Delta_j\|^2 \right\}, \quad (4)$$

where $i \to j$ denotes the indices of the $n - c - 2$ closest updates to Δ_i based on Euclidean distance [30].

1) Multi-Krum: It extends Krum to iteratively select multiple representative updates, enhancing robustness against more malicious clients by aggregating several consistent updates.

III. Methodology

We introduce a reputation-based model sharing algorithm for decentralized federated learning. In this system, each client maintains its own table, which contains various parameters about other clients, and these parameters are updated dynamically. Additionally, each client performs four steps in every round: local training, requesting models, sending models, and model aggregation. Now, let's discuss these steps in more detail.

A. Parameter table

Every client maintains their own table which contains parameters about other clients in the system. In this table contains the Sharing Rate (Q), N_Requesting Rate (R_N), and Trustworthiness (T). The values of these parameters are changing in each round according to the other clients behavior. (Initially all the values are zero).

1) Sharing Rate (Q): The sharing rate function calculates the sharing rate of each client in the decentralized federated learning system. Specifically, if client C_i is calculating the sharing rate for client C_j, it is defined as the ratio of how many times client C_j has sent its model to client C_i when client C_i requested a model from client C_j. The sharing rate of client C_j to client C_i, denoted as $Q(i,j)$, can be mathematically expressed as:

$$Q(i,j) = \left(\frac{\text{Models Received from } C_j \text{ to } C_i}{\text{Requests from } C_i \text{ to } C_j} \right) \times 100\% \quad (5)$$

This metric helps assess the level of contribution each client provides to the corresponding client in the decentralized peer-to-peer learning system.

2) N_Requesting Rate (R_N): The rate calculates how frequently each client requests to send a model. Specifically, if client C_i is calculating the request rate for client C_j, it is defined as a function the ratio of how many times client C_j has requested client C_i's model to the total number of rounds up to the current round. The requesting rate of client C_j to client C_i, denoted as $R(i,j)$, can be mathematically expressed as:

$$R_N(i,j) = \left(1 - \frac{\text{Requests from } C_j \text{ to } C_i}{\text{Number of rounds}} \right) \times 100\% \quad (6)$$

3) N_Own Requesting Rate (O_N): The rate calculates how frequently an own client requests a model from another client. Specifically, if client C_i is calculating the request rate for client C_j, it is defined as a function the ratio of how many times client C_i has requested client C_j's model to the total number of rounds up to the current round. This requesting rate of client C_i from client C_j, denoted as $R(i,j)$, can be mathematically expressed as:

$$O_N(i,j) = \left(1 - \frac{\text{Requests to } C_j \text{ from } C_i}{\text{Number of rounds}} \right) \times 100\% \quad (7)$$

This metric helps assess how greedy each client is in requesting models from others, which can help avoid sending models to highly greedy clients in each round.

4) Trustworthiness Update (T): The trustworthiness score $T(i,j)$ for client C_j as perceived by client C_i is defined as the ratio of the number of times client C_j's model was selected for aggregation of C_i after trimming, to the total number of models received from client C_j. Mathematically, it can be expressed as:

$$T(i,j) = \frac{\text{Number of times } C_j\text{'s models selected by } C_i}{\text{Number of Models Received}_i^j} \quad (8)$$

B. Local Training

Each client i in the federated learning framework maintains a local model $x_t^{(i)}$ at epoch t, trained exclusively on its private dataset D_i. The dataset D_i is partitioned in a *non-IID* manner across clients. During the local training phase, client i refines

its model by minimizing a task-specific loss function $\mathcal{L}$ over D_i. Formally, the local update is computed as:

$$x_{t+\frac{1}{2}}^{(i)} = F\left(x_t^{(i)}, D_i\right) \tag{9}$$

where F represents the local optimization procedure. For standard implementations, F corresponds to iterative stochastic gradient descent (SGD) with τ local epochs:

$$x_{t+\frac{1}{2}}^{(i)} = x_t^{(i)} - \eta \cdot \nabla\mathcal{L}\left(x_t^{(i)}, \xi_i\right) \tag{10}$$

where η is the learning rate, $\nabla\mathcal{L}$ denotes the gradient of the loss function, and $\xi_i \subset D_i$ is a mini-batch sampled from the client's local data.

The intermediate model $x_{t+\frac{1}{2}}^{(i)}$ serves as the client's proposed update, which is subsequently shared or aggregated in later steps of the algorithm. The number of local epochs τ and batch size $|\xi_i|$ are hyperparameters that balance computational efficiency and convergence stability.

Once local training is complete, clients must determine with whom to share their updates to ensure collaborative learning while maintaining robustness and trust. The next step focuses on identifying suitable peers for model exchange.

C. Step 02: Requesting Models

To initiate decentralized collaboration, each client i evaluates potential peers to determine whom to request models from. This is achieved by computing a *requesting score* $P(i,j)$ for every other client $j \neq i$. The score is a function of the Trustworthiness $T(i,j)$, the Own Requesting Rate $O_N(i,j)$, and the Sharing Rate $Q(i,j)$, and is defined as:

$$P(i,j) = [\gamma_T T(i,j) + \gamma_R O_N(i,j)] \cdot Q(i,j)/100, \quad \gamma_T + \gamma_R = 1 \tag{11}$$

Here, γ_T and γ_R are hyperparameters that control the importance of trustworthiness and request behavior. This combined metric ensures that clients prefer peers who are both trustworthy (high $T(i,j)$) and have not been excessively requested by them in the past (high $O_N(i,j)$). The inclusion of $Q(i,j)$ further emphasizes preference for peers with a strong history of cooperation.

Each client then ranks all other clients by descending order of $P(i,j)$:

$$\text{Sorted Clients}^{(i)} \leftarrow \underset{j \neq i}{\text{argsort}} \left(P(i,1), \ldots, P(i,n)\right)_{\text{desc}}.$$

From this sorted list, client i selects the top r_i clients, where r_i denotes its **requesting capacity**—the maximum number of clients it can request models from in a single round:

$$\text{Requesting List}^{(i)} \leftarrow \text{Sorted Clients}^{(i)}[0 : r_i - 1].$$

Each client's requesting capacity r_i depends on factors such as its bandwidth, model size, and aggregation frequency. This step ensures that requests are directed toward peers with the highest expected benefit, improving the likelihood of receiving valuable and cooperative model updates.

Once the requests are sent, clients transition to the model sending phase, where recipients evaluate and decide whether to respond to incoming requests.

D. Step 03: Sending Models

After receiving model requests, each client i evaluates the quality of its requesters to prioritize outgoing transmissions. To ensure fairness and efficient resource allocation, each client assesses the *sending score* $S(i,j)$ of those who requested its model, which is a function of Trustworthiness $T(i,j)$, Sharing rate $Q(i,j)$ and the N_Request rate $R_N(i,j)$ and defined as

$$S(i,j) = [\beta_T T(i,j) + \beta_R R_N(i,j)] \cdot Q(i,j)/100, \quad \beta_T + \beta_R = 1 \tag{12}$$

Where β_R and β_T are hyperparameters. This combined metric ensures that clients prioritize requesters who have both a history of reciprocation (high $Q(i,j)$) and a moderate, non-excessive request frequency (controlled by $R(i,j)$). Clients are then sorted in descending order of $S(i,j)$:

$$\text{Sorted Clients}^{(i)} \leftarrow \underset{j \in \text{Requested Clients}}{\text{argsort}} \left(S(i,1), \ldots, S(i,n)\right)_{\text{desc}}.$$

From this sorted list, client i selects the top s_i clients, where s_i denotes its **sending capacity** (the maximum number of models it can transmit per epoch):

$$\text{Sending List}^{(i)} \leftarrow \text{Sorted Clients}^{(i)}[0 : s_i - 1].$$

Similar to r_i, each client has a unique sending capacity s_i, which depends on its available bandwidth, computational resources, and communication constraints. This step ensures that clients allocate their bandwidth efficiently by prioritizing recipients who have demonstrated both collaborative behavior and fair request patterns. Once the models are successfully transmitted, clients proceed to the critical stage of integrating these external updates into their own training process.

E. Step 04: Model Aggregation

Having received model updates from trusted peers, each client must now integrate these updates while mitigating the effects of non-IID data and unreliable contributors. The aggregation phase refines the local model by incorporating received updates selectively. Let $V = \{x_{t+\frac{1}{2}}^{(j)}\}_{j=1}^{n}$ denote the set of models received from collaborators. The process unfolds as follows:

1) **Trimming**: The outliers are trimmed using a version of Krum aggregation called Multi-Krum (β percentage of models will be removed, $0 \leq \beta \leq 1$):

$$\text{V}_{\text{Trimmed}} = \text{Krum}(V)_{\beta} \tag{13}$$

This eliminates the effect of malicious clients' updates as well as outliers. (Note that, any other outlier model detection algorithm such as [5] can be used for this step)

2) **Weighted Aggregation**: The weighted average is then taken with the client's own model and other selected models, where the weights are determined by the accuracy of each model on the i^{th} client's randomly selected subset of its training dataset.

$$x_{t+1}^{(i)} = \frac{a_i x_{t+\frac{1}{2}}^{(i)} + \sum_{k \in \text{Selected Clients}} a_k V_k}{a_i + \sum_{k \in \text{Selected Clients}} a_k} \tag{14}$$

By carefully selecting and weighting received models, each client refines its local parameters while preserving robustness to non-IID challenges. This completes one iteration of decentralized federated learning, allowing the process to repeat in subsequent rounds with improved global convergence.

IV. Experiment and Results

A. Experimental Setup

To evaluate the proposed approach, we conduct experiments using a simple convolutional neural network (CNN) architecture consisting of two convolutional layers, one max-pooling layer, and two fully connected layers. The experiments are performed on the CIFAR-10 and CIFAR-100 datasets under two different federated learning configurations: one with 20 clients and another with 50 clients.

In the 50-client configuration, 10 clients are designated as selfish, while in the 20-client configuration, 5 clients are selfish. For all experiments, benign clients are assigned a fixed sharing capacity of $s = 5$, and the requesting capacity for all clients is set to $r = 8$. The sharing capacity of selfish clients is fixed within each experiment, but varies across experiments, taking values from 0 to 4.

When reporting results such as model accuracy and the number of received models, we compute and report the average values separately for benign clients and selfish clients, rather than presenting individual client results.

B. Results

Figure 1 shows how the average test accuracy of benign and selfish clients varies with the sending capacity of selfish clients, under both the 20-client and 50-client configurations on the CIFAR-10 dataset. As observed, the average test accuracy of selfish clients increases as their sending capacity increases. Meanwhile, benign clients consistently achieve higher average test accuracy compared to selfish clients across all configurations. Figure 2 presents the results of the same experiments conducted on the CIFAR-100 dataset. The overall trends are similar to those observed with CIFAR-10. However, the test accuracies of benign clients remain more stable across different levels of selfish client sending capacity.

Figure 3 illustrates how the average number of received models for benign and selfish clients changes with the sending capacity of selfish clients, under both the 20-client and 50-client configurations on the CIFAR-10 dataset. Benign clients consistently receive a higher number of models on average compared to selfish clients. However, the average number of

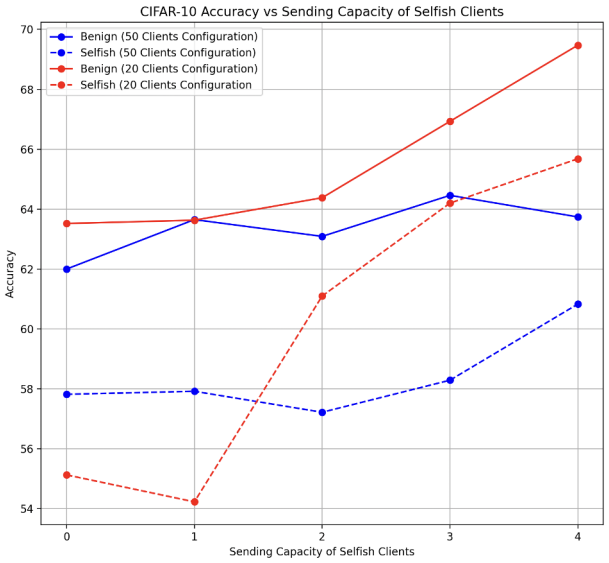

Fig. 1. Average test accuracy for CIFAR-10 under varying sending capacities of selfish clients, comparing scenarios with 50 and 20 total clients, and distinguishing between benign and malicious clients.

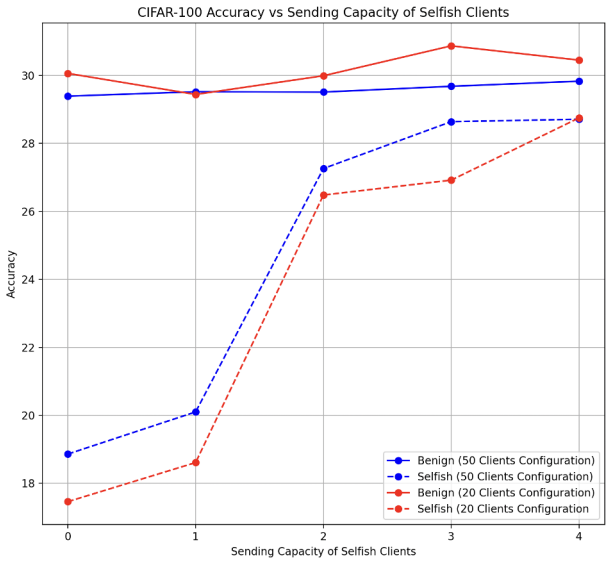

Fig. 2. Average test accuracy for CIFAR-100 under varying sending capacities of selfish clients, comparing scenarios with 50 and 20 total clients, and distinguishing between benign and malicious clients.

received models for selfish clients increases as their sending capacity increases.

The above results provide evidence of the effectiveness of our algorithm, where benign clients gain greater benefits, while selfish clients receive comparatively lower benefits.

C. Ablation Study

In the ablation study we are going to see the effect of the trustworthiness (T) and the sharing rates (R_N and O_N).

TABLE I
COST SUMMARY OF THE PROTOCOL

Metric	Baseline [31]		Ours
	Client	Leader	Client
Computation	$O(n)$	$O(mn^2)$	$O(s^2)$
Communication	$O(n+m)$	$O(n^2)$	$O(s)$
Storage	$O(n+m)$	$O(n+m)$	$O(s)$

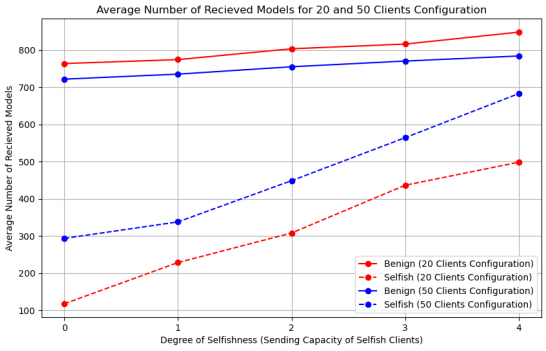

Fig. 3. Average number of received models for benign and selfish clients under 20-client and 50-client configurations, across varying degrees of selfishness (represented by the sending capacity of selfish clients from 0 to 4)

Figure 4 illustrates the impact of trustworthiness on the performance of the algorithm. In this experiment, we vary the value of β_T from 0 to 0.5 to observe the resulting behavior. (Note that we keep $\beta_T = \gamma_T$ throughout all experiments.) As shown in the figure, increasing the weight assigned to trustworthiness leads to improved performance—benign clients receive more models on average, while selfish clients receive progressively fewer models.

D. Algorithm Efficiency

To the best of our knowledge, [31] is the only existing work that explores this topic without adopting a blockchain approach, and we will use it as our baseline. As reported in this baseline paper (Table I), our algorithm exhibits superior efficiency in terms of communication and storage costs. Regarding computational complexity, the baseline method has a cost of $O(n)$ for clients and $O(mn^2)$ (where m is the number of malicious clients and n is the total number of clients). In contrast, our algorithm achieves a computation cost of $O(s)$, where s is the sharing capacity and it is significantly smaller than n. Furthermore, since our algorithm operates solely with clients (without leaders), the average computation cost is lower in most cases compared to the baseline.

V. CONCLUSION

In this paper, we have introduced a novel reputation-based sharing scheme for decentralized federated learning (DFL) that effectively addresses the challenges posed by selfish and malicious clients. Our approach incentivizes cooperation by rewarding clients who contribute meaningfully to the learning process, while deprioritizing selfish clients who exploit the system without providing updates. Moreover, our adaptive reputation mechanism mitigates the impact of adversarial behavior, ensuring that malicious updates do not disrupt the training process. Unlike previous studies, we incorporate clients' resource constraints into the reputation scheme, providing a more realistic model of client behavior in DFL environments.

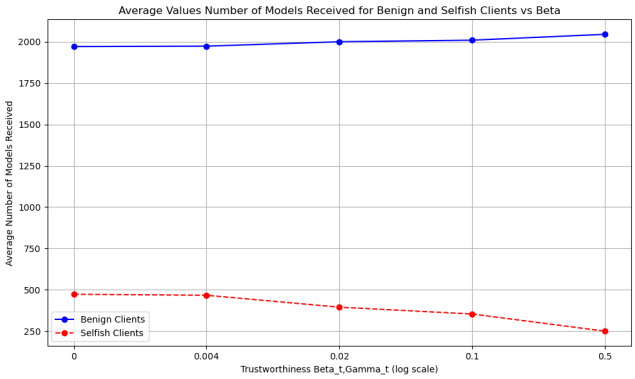

Fig. 4. Average number of received models for benign and selfish clients under 20-client and 50-client configurations, across varying degrees of selfishness (represented by the sending capacity of selfish clients from 0 to 4)

Experimental results demonstrate that our method successfully improves the overall performance of the system by promoting reciprocal behavior among clients and minimizing the influence of non-cooperative participants. Clients with lower contributions consistently receive fewer models, and those exhibiting selfish behavior show significantly poorer testing accuracy. These findings validate the effectiveness of our approach in enhancing both the robustness and sustainability of decentralized federated learning.

VI. FUTURE DIRECTIONS

Future work will focus on further refining adaptive aggregation techniques to tailor them to varying client reliability levels, ensuring robust model performance despite selfish or malicious behaviors. Additionally, strategies will be investigated to improve scalability and communication efficiency, reducing computational costs as the number of clients in decentralized federated learning increases. The system's robustness will be enhanced to counter evolving adversarial attacks, such as backdoor attacks and data poisoning. Moreover, real-world deployment in practical federated learning scenarios, particularly in edge computing environments with resource-constrained devices, will be explored to test the system's practical applicability and performance.

REFERENCES

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

[2] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[3] R. Dai, L. Shen, F. He, X. Tian, and D. Tao, "Dispfl: Towards communication-efficient personalized federated learning via decentralized sparse training," *arXiv preprint arXiv:2206.00187*, 2022.

[4] S. Guo, T. Zhang, H. Yu, X. Xie, L. Ma, T. Xiang, and Y. Liu, "Byzantine-resilient decentralized stochastic gradient descent," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 6, pp. 4096–4106, 2021.

[5] L. He, S. P. Karimireddy, and M. Jaggi, "Byzantine-robust decentralized learning via clippedgossip," *arXiv preprint arXiv:2202.01545*, 2022.

[6] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," *Advances in neural information processing systems*, vol. 30, 2017.

[7] D. Pasquini, M. Raynal, and C. Troncoso, "On the (in) security of peer-to-peer decentralized machine learning," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 418–436.

[8] L. Fowl, J. Geiping, W. Czaja, M. Goldblum, and T. Goldstein, "Robbing the fed: Directly obtaining private data in federated learning with modified models," *arXiv preprint arXiv:2110.13057*, 2021.

[9] D. Pasquini, D. Francati, and G. Ateniese, "Eluding secure aggregation in federated learning via model inconsistency," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2429–2443.

[10] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, S. L. Bernal, G. Bovet, M. G. Pérez, G. M. Pérez, and A. H. Celdrán, "Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 4, pp. 2983–3013, 2023.

[11] E. M. El-Mhamdi, S. Farhadkhani, R. Guerraoui, A. Guirguis, L.-N. Hoang, and S. Rouault, "Collaborative learning in the jungle (decentralized, byzantine, heterogeneous, asynchronous and nonconvex learning)," *Advances in neural information processing systems*, vol. 34, pp. 25 044–25 057, 2021.

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

[13] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, "Fully decentralized federated learning," in *Third workshop on bayesian deep learning (NeurIPS)*, vol. 12, 2018.

[14] Z. Liu, X. Zhang, P. Khanduri, S. Lu, and J. Liu, "Prometheus: taming sample and communication complexities in constrained decentralized stochastic bilevel learning," in *International Conference on Machine Learning*. PMLR, 2023, pp. 22 420–22 453.

[15] X. Zhang, M. Fang, Z. Liu, H. Yang, J. Liu, and Z. Zhu, "Net-fleet: Achieving linear convergence speedup for fully decentralized federated learning with heterogeneous data," in *Proceedings of the Twenty-Third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2022, pp. 71–80.

[16] M. Fang, Z. Zhang, Hairi, P. Khanduri, J. Liu, S. Lu, Y. Liu, and N. Gong, "Byzantine-robust decentralized federated learning," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2874–2888. [Online]. Available: https://doi.org/10.1145/3658644.3670307

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

[18] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "Fltrust: Byzantine-robust federated learning via trust bootstrapping," *arXiv preprint arXiv:2012.13995*, 2020.

[19] K. Kumari, P. Rieger, H. Fereidooni, M. Jadliwala, and A.-R. Sadeghi, "Baybfed: Bayesian backdoor defense for federated learning," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 737–754.

[20] H. Mozaffari, V. Shejwalkar, and A. Houmansadr, "Every vote counts:{Ranking-Based} training of federated learning to resist poisoning attacks," in *32nd USENIX security symposium (USENIX Security 23)*, 2023, pp. 1721–1738.

[21] C. Xie, M. Chen, P.-Y. Chen, and B. Li, "Crfl: Certifiably robust federated learning against backdoor attacks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 372–11 382.

[22] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, "Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients," in *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, 2022, pp. 2545–2555.

[23] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, pp. 1–19, 2019.

[24] L. Witt, M. Heyer, K. Toyoda, W. Samek, and D. Li, "Decentral and incentivized federated learning frameworks: A systematic literature review," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3642–3663, 2022.

[25] Y. Liu, K. Li, Y. Jin, Y. Zhang, and W. Qu, "A novel reputation computation model based on subjective logic for mobile ad hoc networks," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 547–554, 2011.

[26] J. Kang, Z. Xiong, D. Niyato, D. Ye, D. I. Kim, and J. Zhao, "Toward secure blockchain-enabled internet of vehicles: Optimizing consensus management using reputation and contract theory," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 3, pp. 2906–2920, 2019.

[27] M. Shayan, C. Fung, C. J. Yoon, and I. Beschastnikh, "Biscotti: A ledger for private and secure peer-to-peer machine learning," *arXiv preprint arXiv:1811.09904*, 2018.

[28] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Communications Letters*, vol. 24, no. 6, pp. 1279–1283, 2019.

[29] Y. Tian, Z. Guo, J. Zhang, and Z. Al-Ars, "Dfl: High-performance blockchain-based federated learning," *Distrib. Ledger Technol.*, vol. 2, no. 3, Sep. 2023. [Online]. Available: https://doi.org/10.1145/3600225

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

[31] O. Chakraborty and A. Boudguiga, "A decentralized federated learning using reputation," *Cryptology ePrint Archive*, 2024.