Robust and Energy Efficient Client Selection in Federated Learning using Dynamic Gradient Filtering

Francesco Colosimo, Floriano De Rango
Department of Informatics, Modeling, Electronics and Systems (DIMES)
University of Calabria
Rende, Italia
francesco.colosimo@dimes.unical.it, derango@dimes.unical.it

***Abstract*—Federated Learning (FL) presents a novel methodology that has the potential to improve privacy and security when compared to existing methods. This is achieved through allowing numerous users to collaboratively train a machine learning model without sharing the personal data. Notwithstanding its merits, this paradigm could be exposed to poisoning attacks, where an adversary alters local model updates and inserts them into the aggregation process. There are many methods in the literature to ensure robustness against possible attackers, however, the resource-unreliable nature of clients is often not taken into account. In this paper we propose a Robust and Energy Efficient Client Selection (REECS) algorithms that provide an effective resilience against poisoning attacks. In particular, REECS calculates a score for each local model received and associates it with the residual energy of the corresponding client. This process is used to temporarily eliminate from the aggregation the less performing clients with less available energy resources. The attained results showcase the effectiveness of the solutions in terms of both efficiency and resilience.**

***Index Terms*—Federated Learning, Distributed Learning Security, Byzantine attack, Energy-efficient Learning**

I. Introduction

In the last years, several Machine Learning (ML) techniques have been developed to leverage the large amount of data produced every day by mobile and IoT devices both in industry and academia. Many contemporary ML solutions prioritize centralized methods, in which a server gathers all the data for training a robust model. Nonetheless, transferring such extensive data from client devices to the server is frequently impractical due to issues like latency, bandwidth limitations, power constraints, or restrictions imposed by privacy concerns [1]. To overcome these problems, Federated Learning (FL) [2] was proposed. In FL, clients collaboratively train a model under the orchestration of a cloud server while keeping the local training data private (Figure 1). The outcome is the development of a shared and resilient ML model that can effectively tackle important challenges including data privacy, data security, and data access rights [3]. Because of its remarkable problem-solving capabilities, FL is nowadays widely used in various applications, including intrusion and anomaly detection for IoT and visual recognition [4]; the integration of FL into 6G communication also significantly enhances privacy-sensitive applications and unlocks substantial potential for AI-enabled 6G [5].

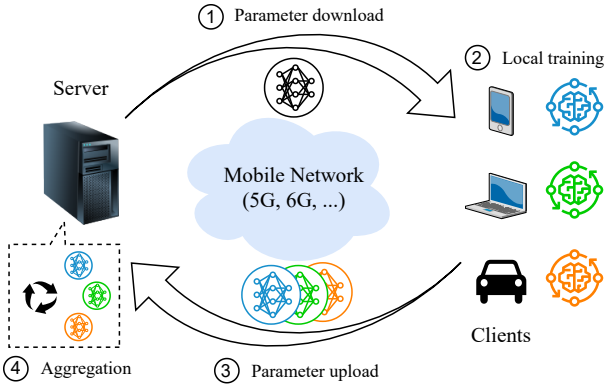

Fig. 1: Illustration of a FL training process that can be divided into four main steps: clients download the global parameter from the server, local training on the clients, local parameter upload to the server, and parameter aggregation performed by the server.

However, due to the unreliable nature of client devices with limited power and bandwidth, and which may exhibit byzantine faults or attacks, it is essential to develop secure and efficient learning techniques. Many previous studies on distributed learning often focus on an optimal situation where all participants function flawlessly and adhere strictly to a predetermined algorithm [6] or do not take into account the possible limitations of edge devices [7]. Therefore, a reasonable goal is to devise an energy efficient distributed algorithm facilitating all the fault-free clients to learn a mathematical model that optimally corresponds to the data points exclusively gathered by the non-faulty agents.

In this paper we first analyze the threat and adversarial model of malicious clients in FL under two different poisoning attacks, namely model poisoning and data poisoning attacks, and discover a valuable pattern that improves the distinction between malicious and honest updates. Therefore, starting from an analysis of the behaviour of Byzantine adversaries and

of the gradient aggregation rules (GAR) that are robust against them [7], [8], we propose a Robust and Energy Efficient Client Selection (REECS) algorithm to screen out malicious behaviors before aggregating model updates to get a new global model, resulting in the development of a robust FL learning framework with the ability to withstand attacks of diverse natures and magnitudes. Finally We provide empirical evidence to demonstrate the effectiveness of this algorithm for distributed learning on neural networks. We run experiments on a benchmark classification task using MNIST dataset, under a model poisoning and a data poisoning attack. The results demonstrate that this approach is energy efficient and competitive to maintain robustness against Byzantine adversaries while mitigating data heterogeneity.

II. Related works and Problem setting

In the broad area of secure and efficient distributed learning over networks, several works have appeared in the literature since Google's first FL implementation [2], where the authors proposed the *Federated Averaging* (*FedAvg*), an FL algorithm that has been extensively applied as it is simple to implement. While this strategy is useful, it does not provide a guarantee of resilience against Byzantine faults or attacks.

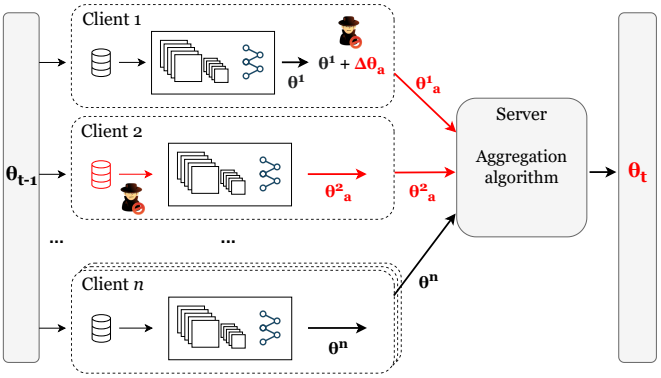

Fig. 2: Federated environment where poisoning attackers are present. In particular, *Client 1* represents a model poisoning attacker that alters the trained parameters by adding them to values randomly sampled from a normal distribution. *Client 2* depicts a data poisoning attacker who performs a label-flip on local data.

In [7], Blanchard et al. present *Krum* and *Multi-Krum*, two aggregation methodologies robust to such attacks. In this algorithms the central server selects only the update (*m* updates in *Multi-Krum*) that is closest to its neighbors and discards all other updates. On the other hand, *FoolsGold* [8] detects attacks by analyzing the diversity of client updates over the FL process using a *cosine similarity* technique. Xie et al. [9] propose three aggregation rules: geometric median, marginal median, and mean around median. The first creates a new update that minimizes the summation of the distances among each local update; the other two are based on the one-dimensional median and on a trimmed average respectively. In contrast, ours previous work [10] [11] proposes a novel aggregation algorithm that exploit the capabilities of both distance and statistics approaches, thus "selecting a subset of models that are as close as possible to the other updates and analysing their statistical properties in order to discard the abnormal models".

However, most of the approaches listed above [7] [8] are only efficient in particular attack scenarios, as they depend on specific assumptions regarding the attack or the data distribution and often do not take into account the limited resources of edge devices (the selection of clients is at most made casually). On the other hand, [6] relies on a greedy algorithm with a knapsack constraint, selecting as many clients as possible within a predefined deadline. In [12], the authors present an algorithm that adaptively discovers a client selection strategy that is dependent on the availability of resources, while asymptotically reducing the influence of client-sampling variance on the convergence of the global model. In [13] the proposed algorithm removes adverse local updates by comparing the gradients of the local and the global model, and thus selects clients that accelerate the convergence of the model. It is therefore worth noting that there are not many works that focus on robustness against possible adversaries and at the same time try to preserve the limited client resources.

III. Threat model: Poisoning attack

Recent investigations have shown that the standard FL strategy can be undermined by Byzantine attacks carried out by participants who are either mistaken or malicious [14]. Training on client devices poses a risk of clients manipulating training data or disrupting the convergence process by tampering with uploaded data [15]. In some cases, a single client can completely nullify the accuracy of the model.

In the context of this study, client attacks can be classified as model poisoning attacks, which aim to degrade the accuracy and correctness of the learning process during the training phase, and into data poisoning attacks, wherein the attacker compromise the integrity of datasets [8]. Figure 2 shows a federated environment with an example of model and data poisoning attacks.

A. Adversarial capability: model poisoning attack

An example of model poisoning attacks is the *Gaussian* attack, which randomly modifies the local models of compromised clients. In detail, a *Gaussian* attacker alters the trained parameters by adding them to values randomly sampled from a normal distribution. Eq. 1 shows the mathematical model of *gaussian* attack,

$$\theta_i^{t+1} = \begin{cases} \theta^t - \eta \nabla L(\theta^t, D_i), & no\ attack \\ \theta^t - \eta \nabla L(\theta^t, D_i) + N(\mu, \delta^2)^d, & gaussian \end{cases} \quad (1)$$

where θ_i^{t+1} denotes client i gradient in the tth +1 iteration, D_i a local dataset, $N(\mu, \delta^2)$ a normal distribution with μ mean and δ standard deviation.

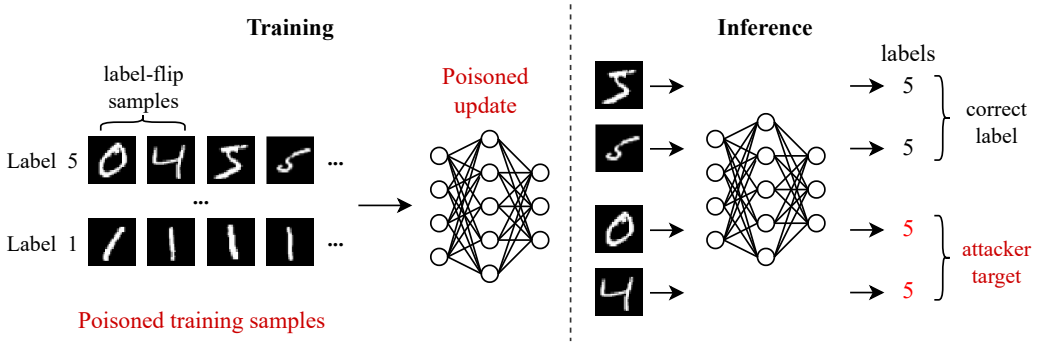

Fig. 3: Illustration of a typical data poisoning attack called label-flipping attack.

B. Adversarial capability: data poisoning attack

Data poisoning attacks, as already stated, involve the attacker compromising the integrity of local datasets. One of the most common examples is the *label-flipping* attack [16], in which the adversary alters the labels of its training data, flipping either some or all of them to a target class or arbitrary ones, while maintaining the input data features unchanged. After poisoning the data, the attacker trains local model with them as follows:

$$\theta_i{}^{t+1} = \theta^t - \eta \nabla L(\theta^t, D_{lf}) \tag{2}$$

where $\theta_i{}^{t+1}$ denotes the malicious client i update in the tth +1 iteration and D_{lf} a label flipped dataset. Because attackers can only manipulate a subset of the labels and might intermittently conceal their malicious behavior with unaltered data, detecting the label-flipping attack can prove challenging for the server, which also lacks access to the raw data of the clients. The illustration in Figure 3 depicts a label-flipping attack. In this example, label 0 and label 4 are flipped to label 5 and fed to the learning algorithm. If unmitigated, the FL aggregation process includes this (incorrect) contribution, resulting in a misclassification during the inference process.

IV. Algorithm Design

The approaches and techniques for addressing potential threats vary depending on the fundamental principles employed by the server to identify or prevent abnormal updates. Existing defense mechanisms rely on either robust aggregation rules based on gradient distances or statistics to ensure the accuracy of gradient estimations [7], [10]. Other methods require integrating more data into central servers to verify the gradients received [17]. In both situations, the aggregation rule is expected to produce a vector that closely corresponds to the actual gradient, specifically indicating the direction of the steepest ascent of the cost function undergoing optimization. In this paper we aim to create a defense system that is effective and possesses the following characteristics:

- Robustness: Our objective is to establish a defense mechanism that efficiently thwarts the adversary's malicious intentions by precisely detecting and neutralizing as many harmful updates as possible, thereby minimizing their impact on the global model.
- Efficiency: The global model's performance must be maintained at a high level in order to preserve its usefulness, while trying to minimise resource consumption by edge devices.

A. Filtering adverse local update

Following a majority-based approach, it is possible to highlight the relationships between gradients by measuring the distances between them and assigning each local update a value equal to the sum of the distances to the other updates [7], as shown in the following equation:

$$s_i = \sum_{i,j \in n} \|\theta_i - \theta_j\|^2, \; i \neq j \tag{3}$$

Considering that small magnitudes of gradients are considered less damaging to training, while a significantly large one is definitely malicious, we can set a threshold equal to the average of the clients' distance scores to delineate which gradients to consider for aggregation: in a straightforward sense, only gradients whose score is below the average can contribute to the global model, i.e.

$$\theta^{t+1} = GAR(\theta^t_{i \in P'}) \tag{4}$$

where $P' = [i : s_i \leq avg\_s,]$ is the set of selected clients whose score (Eq. 3) fall within the threshold $avg\_s = avg\{s_k : k \in [n]\}$ and with which to perform the aggregation.

B. REECS algorithm

Based on statements made above, in this paper we propose REECS, a novel energy efficient client selection with robust aggregation algorithm that can dynamically filter out underperforming and malicious clients, while trying to maximise the lifespan of honest clients (Algorithm 1). Unlike algorithms such as [7] that require the number of malicious clients in advance, our proposal updates the number f of adversaries based on the difference between the total number of participants and the cardinality of the P subset of clients deemed honest. The steps characterising this approach are outlined below.

Algorithm 1 Energy efficient client selection and robust aggregation algorithm with Dynamic Gradient Filter.

Server:
Initialize global model θ^0 and the constant ℓ
Initialize the number of adversaries $f = 0$
for each training round *t = 1, 2, ...T* **do**
  if $t\%\ell$ is equal to 1 **then**
    Select $n = C \times K$ clients, where $C \in (0, 1)$
  else
    Select $n = P$ clients
  for each **Client** *k = 1, 2, ...n* **in parallel do**
    Download θ^t to Client k
    Do **Client** k update and receive θ_k, E_k
  if $t\%\ell$ is equal to 1 **then**
    for each **Update** *i = 1, 2, ...n* **do**
      Compute a score s_i as the sum of the distances between i and the nearest $n - f - 1$ nodes
    Select a set P of models with s_i lower than $avg\{s_i : i \in [n]\}$ and with a higher E_k
    Update $f = n - p$, with $p = |P|$
  Update global model $\theta_t \leftarrow median\left\{\theta^i : i \in P\right\}$

Client k update:
Replace local model $\theta_k \leftarrow \theta^t$
for local epoch from 1 to E **do**
  for batch $b \in$ (1,B) **do**
    $\theta_k \leftarrow \theta_k - \eta \nabla L(\theta_k, b)$
Calculate residual energy E_k
return θ_k, E_k

Step 1. After defining every how many iterations to check the behaviour of the clients and then update the selection of participants for the next rounds (ℓ parameter), following the FL paradigm, at each iteration REECS receives the various local parameter vectors (Figure 4a). In addition to the latter, REECS also requires clients to send in their residual energy.

Step 2. The algorithm then analyses the behaviour of the clients according to the distance score obtained by each of them (Eq. 3). Using the filtering threshold mechanism described above (Eq. 4), REECS selects a subset *P* of clients that are evaluated as well-performers and non-malicious. The parameter f of possible attackers (equals to 0 in the first iteration) is then calculated as the difference between the complete set of clients and *P*. This allows REECS to better adapt to possible variations in client behaviour compared to algorithms in which the number of attackers must be statically declared a priori.

Step 3. Let S and E be the score and residual energy vectors ordered by arrival at the server. Having calculated the index arrays χ_1 and χ_2 that would sort S and E in ascending and descending order respectively, these indices are merged to obtain a new arrangement that takes both aspects into account. In detail, for each $i = 0, ..., len(S) - 1$, REECS creates a new

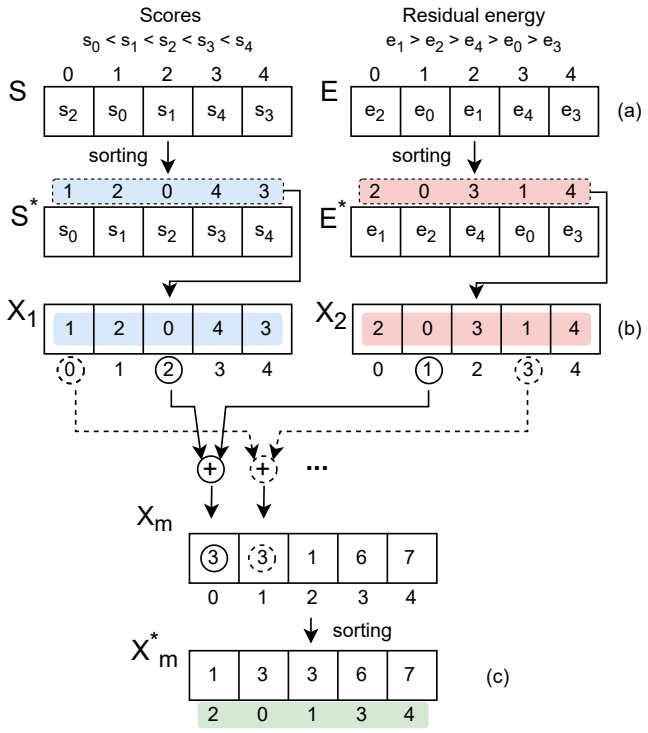

Fig. 4: REECS client selection.

vector χ_m following the formula

$$\chi_m[i] = index(i, \chi_1) + index(i, \chi_2) \tag{5}$$

i.e. each of χ_m positions i contains the sum of the index of the element i in χ_1 and in χ_2. Referring to the example in Figure 4, the first element ($i = 0$) of χ_m is calculated from the sum of the index of element 0 in χ_1 (index 2) and in χ_2 (index 1), i.e. $\chi_m[0] = 2 + 1 = 3$. In this way, the elements in χ_m faithfully respect the order of arrival of the client parameters in S and E, however, each of the clients is associated with a new value obtained by merging the positioning of the same client in the two metrics. Ordering χ_m (namely χ^*_m) will therefore have in first position the client whose performance is better overall. The algorithm then update P with the first $n-f$ clients in χ^*_m (Figure 4c).

Step 4. REECS then continues for the next ℓ rounds by selecting only those clients belonging to P and thus proceeds to aggregate the related parameter vectors using the statistical *geometric median* approach; the use of this methodology is necessary to ensure the robustness of the algorithm in the event of estimation errors in the filtering. The ℓ rounds ended, it reconsiders all available clients again and resumes from *Step 1*, all until the desired number of rounds has been reached.

V. Experiments

A. Setup

The experiments reported in this article are conducted on a 92 core cluster with 512 GB of RAM and follow the standard FL scheme, i.e. a distributed context comprising a

central server and $n = 20$ client nodes, among which up to f malicious adversaries may be present. All federated nodes are implemented with Flower framework [18] inside virtual machines and imported into GNS3 software emulator in order to realistically simulate a distributed environment over networks. The number of adversaries may vary over time and the adversaries can exhibit a model poisoning called Gaussian attack, in which the parameters are perturbed by adding values sampled from a normal distribution with mean 0.0 and standard deviation equal to 20, and a label-flipping attack, where the original label of a data point y sampled by all malicious clients is changed to $\tilde{y} = 9 - y$. The algorithms selected to perform this analysis are the REECS (with $\ell = 5$) and the well-known *FedAvg* [2], *GeoMed* [9], *Krum* and *Multi-Krum* [7].

Hyperparameters		Setup	
Optimizer	SGD	# Clients	20
Learning rate	0.001	Neural network	LeNet-5
Momentum	0.9	# parameters	60 M
Epochs	1	Bandwidth	100 Mbps
Batch size	32	Client energy	0.013 kWh
# rounds	50	ℓ	5

TABLE I: Experiments configuration.

Regarding the dataset, the image classification collection of handwritten digits MNIST is used. In our experiments, it was agreed to split the trainset into twenty parts, giving one to each client. Moreover, this division is carried out using a Latent Dirichlet Allocation to get a no-IID and imbalanced distribution. We also use a state-of-the-art *LeNet* neural network [19] with five layers and about 60 thousand parameters.

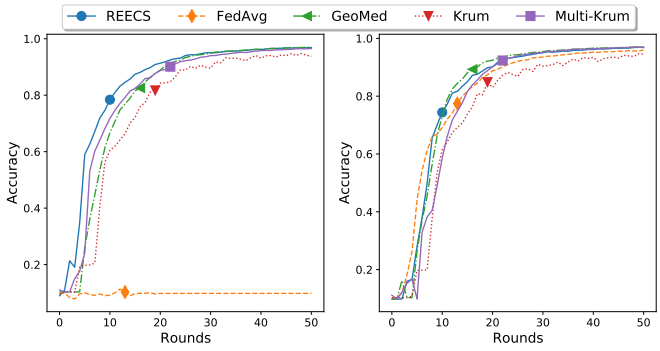

Fig. 5: Distributed learning of LeNet-5 for MNIST with different aggregation algorithms (represented by different colors). In this figure the testing accuracies evaluated by the server after 0 to 50 steps of the different algorithms in precence of $f \leq 20\%$ gaussian attackers (left) and label-flipping attackers (right) are plotted.

B. Results: poisoning attack

An initial test investigated the behaviour of the cited aggregation algorithms in presence of Gaussian and label-flipping attacks, and varying over time the number of adversaries (f up to 20%). Furthermore, we maintain consistency during the training phase by establishing fixed random seeds that agents utilize to sample data. This ensures that in different experiments, the same agent will choose the same mini-batch of data for a specific iteration. From the graph shown in Figure 5, it can be seen that, under the hypothesis of *Gaussian attacker* (right side) it can be observed that all algorithms, except for *FedAvg* (previously demonstrated to lack tolerance in the literature), exhibit robustness against such an attack, as they all attain satisfactory results. The *FedAvg* instead manages to converge in the case of the *label-flipping* attack because its weighted average manages to mitigate the lighter impact of this attack on the global model.

C. Energy efficiency

Given the decentralized nature of FL and the common use of IoT devices as clients, it is essential to evaluate the resource utilization of each algorithm. In this context, a high number of communication rounds implies a higher energy usage, which may not always be permissible for such devices. Building on the research conducted by [20], the client energy consumption can be simplified in this context using the formula $E_{FL} = E_{train} + E_{comm}$, with:

$$E_{train} \simeq T \cdot \tau \cdot W \tag{6}$$

$$E_{comm} \simeq 2 \cdot \sum_{i=1}^{T} N_c^i \cdot \frac{W_c}{R_c} \tag{7}$$

where T is the total number of rounds, τ and W represent the training time and the training power per round, W_c, R_c and N_c are, respectively, the consumed power in communication, the bit rate and the size of transmitted data. For simulation purposes, these energy values are estimated at runtime using the CodeCarbon library [21]. The graph in Figure 6 illustrate a FL excecution over 50 rounds where all clients have an initial energy of 0.013 kWh [22]. In it it is worth noting how REECS results in energy savings (from a minimum of 5% to a maximum of 21%) in each of the 20 clients present when compared to traditional FL algorithms such as *Multi-Krum*.

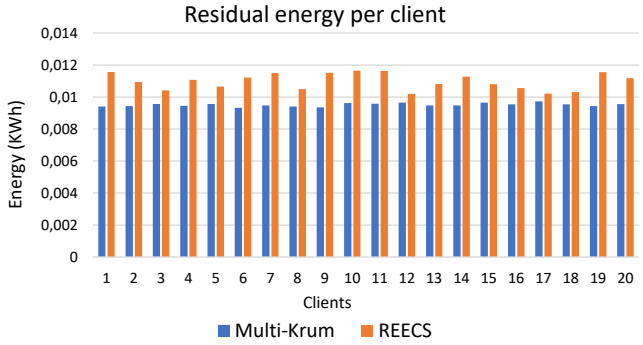

Fig. 6: Analysis of clients residual energy after 50 rounds. In it can be seen how REECS (with ℓ equal to 5) results in energy savings in each of the 20 clients present when compared to traditional FL algorithms such as *Multi-Krum*.

D. Resilience against residual energy poisoning

The way the algorithm is designed, a malicious client could attempt to deceive REECS by falsifying (make it tend to infinity) its residual energy with the intention of having a better chance of belonging to the set of clients whose contribution creates the global model. As previously stated, REECS uses the geometric median for the final aggregation, which is proven to be robust against Byzantine attackers. This means that even when an attacker should then be included in the set P, this function allows for no loss of accuracy. Table II shows the accuracy values at the 50th round carried out considering that a α percentage of gaussian attackers also falsify their residual energy: even when all adversaries falsify their energy, our proposal succeeds in achieving convergence.

REECS (f = 40%)				
	$\alpha = 20\%$	$\alpha = 50\%$	$\alpha = 70\%$	$\alpha = 100\%$
Accuracy (%)	96.8	96.1	96.7	95.1

TABLE II: Mean results for the REECS testing accuracy evaluated by the server after 50 rounds with the presence of 40% of gaussian attacker, α of which also falsify their residual energy.

VI. Conclusions

In this paper, we introduce an energy efficient client selection methodology designed to impart Byzantine fault tolerance to distributed federated learning. Our analysis, comparing the REECS algorithms with well-known approaches, demonstrates that the former is comparable and, in some instances, surpasses the limitations of the latter. Notably, REECS algorithms can better leverage the experience of honest clients by dynamically adjusting the estimated number of opponents at each step. Additionally, this methodology uses information regarding the residual energy of edge nodes in order to maximise the lifetime of each one. This renders it a pragmatic approach for defending against poisoning attacks. Comprehensive experimental results validate both our theoretical insights and empirical findings, highlighting the remarkable efficacy of our proposed approach. While further studies, such as investigating the impact of the constant ℓ, are desirable, in the context presented, the REECS algorithm achieves optimal accuracy compared to existing algorithms.

VII. Acknowledgments

This work was partially supported by project SERICS (PE000 00014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

References

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

[2] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," *Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017. JMLR: W&CP volume 54*, Feb. 2016.

[3] Y. Chen, Y. Gui, H. Lin, W. Gan, and Y. Wu, "Federated Learning Attacks and Defenses: A Survey," Nov. 2022.

[4] Z. Zhang, S. Ma, Z. Yang, Z. Xiong, J. Kang, Y. Wu, K. Zhang, and D. Niyato, "Robust semisupervised federated learning for images automatic recognition in internet of drones," *IEEE Internet of Things Journal*, vol. 10, no. 7, pp. 5733–5746, 2023.

[5] Z. Yang, M. Chen, K.-K. Wong, H. V. Poor, and S. Cui, "Federated learning for 6g: Applications, challenges, and opportunities," *Engineering*, vol. 8, pp. 33–41, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809921005245

[6] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. IEEE, May 2019. [Online]. Available: http://dx.doi.org/10.1109/ICC.2019.8761315

[7] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine Learning with Adversaries: Byzantine Tolerant Gradient Descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 118–128.

[8] C. Fung, C. J. M. Yoon, and I. Beschastnikh, "Mitigating Sybils in Federated Learning Poisoning," Aug. 2020.

[9] C. Xie, O. Koyejo, and I. Gupta, "Generalized Byzantine-tolerant SGD," Feb. 2018.

[10] F. Colosimo and F. De Rango, "Median-krum: A joint distance-statistical based byzantine-robust algorithm in federated learning," ser. MobiWac '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 61–68.

[11] F. Colosimo and F. De Rango, "Distance-statistical based byzantine-robust algorithms in federated learning," in *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*, 2024, pp. 1034–1035.

[12] M. Ribero, H. Vikalo, and G. de Veciana, "Federated learning under intermittent client availability and time-varying communication constraints," *IEEE Journal of Selected Topics in Signal Processing*, vol. 17, no. 1, p. 98–111, Jan. 2023.

[13] H. Wu and P. Wang, "Node selection toward faster convergence for federated learning on non-iid data," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, pp. 3099–3111, 2022.

[14] K. Zhang, X. Song, C. Zhang, and S. Yu, "Challenges and future directions of secure federated learning: a survey," *Frontiers of Computer Science*, vol. 16, no. 5, Dec. 2021.

[15] P. Kairouz and al., "Advances and Open Problems in Federated Learning," Dec. 2019.

[16] N. M. Jebreel, J. Domingo-Ferrer, D. Sánchez, and A. Blanco-Justicia, "Defending against the label-flipping attack in federated learning," 2022.

[17] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping," Dec. 2020.

[18] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A Friendly Federated Learning Research Framework," Jul. 2020.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[20] A. A. Al-Saedi, V. Boeva, and E. Casalicchio, "Reducing communication overhead of federated learning through clustering analysis," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, 2021, pp. 1–7.

[21] "CodeCarbon." [Online]. Available: https://codecarbon.io/

[22] G. Callebaut, G. Leenders, J. Van Mulders, G. Ottoy, L. De Strycker, and L. Van der Perre, "The art of designing remote iot devices—technologies and strategies for a long battery life," *Sensors*, vol. 21, no. 3, 2021.