FLAD: Adaptive Federated Learning for DDoS Attack Detection

Roberto Doriguzzi-Corin, Domenico Siracusa
Cybersecurity Centre, Fondazione Bruno Kessler, Italy

✦

Abstract—Federated Learning (FL) has been recently receiving increasing consideration from the cybersecurity community as a way to collaboratively train deep learning models with distributed profiles of cyber threats, with no disclosure of training data. Nevertheless, the adoption of FL in cybersecurity is still in its infancy, and a range of practical aspects have not been properly addressed yet. Indeed, the Federated Averaging algorithm at the core of the FL concept requires the availability of test data to control the FL process. Although this might be feasible in some domains, test network traffic of newly discovered attacks cannot be always shared without disclosing sensitive information. In this paper, we address the convergence of the FL process in dynamic cybersecurity scenarios, where the trained model must be frequently updated with new recent attack profiles to empower all members of the federation with the latest detection features. To this aim, we propose FLAD (adaptive Federated Learning Approach to DDoS attack detection), an FL solution for cybersecurity applications based on an adaptive mechanism that orchestrates the FL process by dynamically assigning more computation to those members whose attacks profiles are harder to learn, without the need of sharing any test data to monitor the performance of the trained model. Using a recent dataset of DDoS attacks, we demonstrate that FLAD outperforms state-of-the-art FL algorithms in terms of convergence time and accuracy across a range of unbalanced datasets of heterogeneous DDoS attacks. We also show the robustness of our approach in a realistic scenario, where we retrain the deep learning model multiple times to introduce the profiles of new attacks on a pre-trained model.

Index Terms—Network Security, Intrusion Detection, Distributed Denial of Service, Federated Learning, Heterogeneous Data

1 INTRODUCTION

As the number and complexity of cybersecurity attacks increase at a tremendous pace on a daily basis [1], defenders are in need to find more effective protection measures that rely on machine intelligence. To this account, a recent trend in information security is the adoption of solutions based on Artificial Neural Networks (ANNs) to analyse network traffic and the behaviour of software running on computers to identify possible compromised systems or unauthorised access attempts [2], [3]. Compared to traditional signature-based and anomaly-based approaches, ANN-based threat detection methods are more resilient to variations in attack patterns and are not constrained by the requirement to define thresholds for attack detection. However, training and updating an ANN model for effective threat detection is a non-trivial task, due to the complexity and variability of emerging attacks and the lack of data with relevant and up-to-date attack profiles, especially when dealing with zero-day vulnerabilities.

Collaborative learning is a recent approach that addresses the challenges associated with data and ANN model updates. It enables multiple independent parties to train and update their Intrusion Detection System (IDS) by sharing information on recent attack profiles. In this scenario, a security provider could offer an IDS trained on incidents experienced by all its customers, ensuring a service that is continuously updated with the latest attacks. Collaborative learning techniques have started to gain attention in recent years, when McMahan et al. [4] presented the so-called Federated Learning (FL), a distributed training approach with focus on the privacy of the individual participants in the FL process. FL relies on a set of participants (also called clients) that train the model on their local data, and on a central server that aggregates ANN model parameters collected from clients and distributes the aggregated model back to clients for further training sessions. This sequence of operations is executed multiple times (federated training rounds) with no exchange of clients' private training data, until a target convergence level is reached.

The application of FL in cyber security for intrusion detection has been explored in previous research [5], [6], [7]. However, previous works rely on Federated Averaging (FEDAVG), the FL mechanism introduced by McMahan et al., which necessitates a representative test set available at the server side to control the training process. We argue that this approach poses a data privacy issue and may restrict the applicability of FL in scenarios where only a subset of data classes can be tested by the server. It is reasonable to assume that network data containing recent cyber incidents against one or more clients may include sensitive information that cannot be shared with the server for testing purposes. Consequently, in such cases, the server would not have the ability to assess the performance of the aggregated model using the latest attack traffic. Furthermore, achieving convergence in the FL process can present challenges due to several factors. These include the presence of non-independent and identically distributed (non-i.i.d.) data across clients, as well as unbalanced datasets, which are common in network anomaly detection. Slow convergence can hinder the ability to promptly update the IDS service in response to attacks targeted at specific clients within the federation. While some of these issues have been addressed to some extent in previous works, their effectiveness remains

uncertain, as outlined in the subsequent sections.

In this paper, we propose a novel adaptive Federated Learning Approach to DDoS attack detection (FLAD), in which the server verifies the classification accuracy of the global model on clients' validation sets with no exchange of training or validation data, granting that the model is learning from all clients' data and allowing to implement an effective early-stopping regularisation strategy. FLAD is conceived to apply FL in the cybersecurity domain, where we assume that no attack data will be shared at any time between the clients (e.g., customers of an IDS service) and the server (e.g., provider of the service). We tackle the convergence of the federated learning process in the context of Distributed Denial of Service (DDoS) attack detection, with focus on the trade-off between convergence time and accuracy of the merged model in segregating benign network traffic from a range of different DDoS attack types. We consider a dynamic scenario, where clients are targeted by zero-day DDoS attacks, and where the global model must be updated with new information as soon as possible to empower all participants with the latest detection features.

The high-level idea behind FLAD is to involve in a training round only those clients that do not obtain sufficiently good results on their local validation sets with the current global model. For such clients, the amount of computation (number of training epochs and gradient descent steps/epoch) is determined based on their relative accuracy on their validation sets. Note that, the accuracy score is computed by clients on their validation sets and communicated to the server upon request. Hence, no exchange of sensitive data between server and clients is involved. Compared to FEDAVG, FLAD introduces a negligible traffic overhead between clients and server, without disclosing clients' sensitive data, even for testing purposes.

We evaluate FLAD in a worst-case scenario, where the DDoS attack data among the clients is unbalanced and non-i.i.d.. We compare FLAD against FEDAVG and FLDDoS [5], a state-of-the-art DDoS attack detection tool that builds upon FEDAVG and is designed to address the issues associated with non-i.i.d. data. We demonstrate that FLAD improves FEDAVG and FLDDoS in terms of training time, number of training rounds/client and classification accuracy on unseen traffic data. That is, our approach allows for a faster global model update, requires less computation on clients, and ensures a high accuracy on all DDoS attack types.

The main contributions of this work are the following:

- An analysis of the limitations of the FEDAVG algorithm in cybersecurity applications with unbalanced and non-i.i.d. data.
- FLAD, a novel adaptive mechanism that addresses the aforementioned limitations by steering the federated training process in terms of client selection and amount of computation for each client.
- An extensive evaluation on a recent dataset that compares our approach against the FEDAVG algorithm and FLDDoS, demonstrating that FLAD is more efficient and outputs aggregated models of higher classification accuracy.
- A prototype implementation of FLAD, publicly available for testing and use [8].

The remainder of this paper is organised as follows. Section 2 presents the FEDAVG algorithm and highlights its limitations in training models for cybersecurity applications. Section 3 reviews and discusses the related work. Section 4 provides a threat model analysis. Section 5 presents the FLAD adaptive federated training for DDoS attack detection. Sections 6 and 7 detail the dataset and the experimental setup. In Section 8, FLAD is evaluated and compared against state-of-the-art FL solutions. Section 9 analyses the security risks of FLAD and discusses the available techniques to mitigate them. Finally, the conclusions are given in Section 10.

2 PROBLEM FORMULATION

Federated Learning (FL) was introduced in 2017 by McMahan et al. [4] as a communication-efficient process for training neural networks on decentralised data. The paper formulates the FEDAVG algorithm, which is proposed to optimise the federated learning process in real settings, including non-i.i.d. and unbalanced datasets. The FL process involves a central server and a set of K clients, each with a fixed local dataset. Such a process consists of several *rounds* of federated training during which the server selects a random fraction F of clients (for efficiency reasons) and sends them an ANN model for local training. The selected clients train the model with local data and send it back to the server, which integrates all the updates with the global model. This process is iterated for several rounds until the desired test-set accuracy is reached. The key aspects in this process are three: the aggregation of local updates, the amount of computation performed at each round and the training-stopping strategy, where the latter assumes the availability of test data at the server location.

The aggregation of clients' updates is based on the FEDAVG algorithm, formulated in Equation 1, which computes the average of clients' models weighted with the number of local training samples (n_k).

$$w_t \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_t^k \qquad (1)$$

In Equation 1, $n = \sum_{k=1}^{K} n_k$ is the total number of training samples, while w_t^k represents the set of parameters of client k at round t. Please note that the aggregation is always performed using the weights of all K clients, although only a fraction F of them have been updated during round t. For all the other clients, the weights of round $t-1$ are used.

Two main parameters control the amount of computation necessary at each round of the FL process. The fraction of clients F that perform local training, and the number of local updates performed by each client k, which is computed as $u_k = E \cdot S = E \cdot \frac{n_k}{B}$, where E is the number of training epochs and S is the number of gradient descent steps for each epoch, which depends on the batch size B, such that $S = \frac{n_k}{B}$. McMahan et al. report the outcomes of various experiments on image classification and language modelling tasks in terms of communication rounds with different combinations of F, E and B, which are kept constant during each experiment.

2.1 Limitations of FEDAVG

In Section 8, we demonstrate that the FEDAVG algorithm, as conceived by McMahan et al., does not satisfy two basic requirements for effective DDoS attack detection:

1) Short convergence time to reach the target attack detection accuracy, especially in emergency threat situations in which the global model must be quickly distributed to clients upon retraining with recent DDoS attack information. Indeed, FEDAVG assigns the same amount of computation to all the clients selected for a round of training, irrespective of the accuracy level reached by the global model on specific clients' data. This inefficient management can lead to long FL training sessions with no substantial gain in accuracy.

2) Accurate detection of all attack types in realistic conditions, where the detection system must learn from unbalanced and non-i.i.d. data obtained from heterogeneous DDoS attack types characterised by different traffic rates and feature distributions. The weighted average of FEDAVG gives more importance to the weights of the clients with large local training sets, to the detriment of the smallest ones. We argue that this strategy could hinder FEDAVG's ability to detect attacks characterised by out-of-distribution features that are available only in small local training sets.

Furthermore, it should be noted that FEDAVG operates under the assumption that some test data is accessible at the server site to verify that a target accuracy of the global model is achieved and stop the training process. We argue that this assumption rarely holds in the cybersecurity domain. For instance, let us consider a scenario where one client contributes with updates related to zero-day attack traffic that is not public at training time. In this case, the only solution for the server to verify that the model has learned the new attack would be to use the client's test set. However, even if we discount the willingness of the client to provide such information, this would require data cleaning (anonymisation) from the client's sensitive information, with the risk of losing IP, transport and application layer features that could be critical for model validation.

2.2 Problem statement

Our problem of DDoS attack detection in federated environments can be formulated as the maximisation of global model accuracy on unbalanced, non-i.i.d. data across clients, while minimising total FL time. Based on the previous discussion, the solution must satisfy the following constraints:

C1 No training data can be shared among clients or between clients and the server.

C2 No test data is available at the server location.

The solution to the above problem is challenging due to the competing objectives of maximum accuracy and minimum training time and, on the other hand, the limited data (we assume no data at all) available for the server to assess the performance of the global model during the FL process.

3 RELATED WORK

Implementing a robust and efficient FL system is a complex task [9] that often involves domain-specific tuning and optimization. In cybersecurity, recent works have addressed issues related to non-i.i.d. and unbalanced data, with a primary focus on performance aspects such as accuracy and convergence time. Nevertheless, it is important to acknowledge that these works heavily depend on the vanilla FEDAVG algorithm, thereby inheriting the limitations discussed in Section 2.1. In this section, we provide a comprehensive review and discussion of the current state-of-the-art in FL research, with a particular emphasis on challenges specific to the cybersecurity domain.

3.1 Federated Learning in cybersecurity

In cybersecurity, FL methods can be exploited to share attack and anomaly profiles with other parties with no disclosure of private data. In this regard, FedOE [10], LwResnet [7], FIDS [11] and two tools called FLDDoS [5], [6] are recent solutions for DDoS attack detection evaluated on the CIC-DDoS2019 dataset, the same used to assess the performance of FLAD (cf. Section 6). FedOE is a FL-based framwork that resorts on semi-supervised learning to detect DDoS attacks. Clients share with the server the minimum and maximum anomaly scores obtained on their local datasets. The scores are used to find the optimal threshold that maximises the F1 Score across all the clients.

LwResnet is a lightweight residual network that has been evaluated in FL settings with FEDAVG, using a subset of 6 UDP-based attacks out of the 13 attacks available in the CIC-DDoS2019 dataset. The first tool called FLDDoS [5] addresses the challenges associated with non-i.i.d. data by combining FEDAVG with a local weighted average conducted by individual clients. The local average incorporates the global model received from the server, along with a local model that is trained exclusively with the client's local data. Given the similarities with our work, in Section 8.1 we will compare FLAD against FLDDoS in terms of convergence time and accuracy on non-i.i.d. data. The authors of the second solution called FLDDoS [6] propose a methodology that tackles the issue of local data imbalance through data augmentation. In addition, they suggest a two-stage model aggregation approach that helps to reduce the number of federated training rounds. On the other hand, FIDS is proposed to improve the performance of FEDAVG on non-i.i.d. data with feature augmentation. This technique requires sharing a representation of the client's data with the central server. Dimolianis et al. [12] focus on collaborative DDoS attack mitigation using programmable firewalls. In this work, a Multi-Layer Perceptron (MLP) model is trained using FEDAVG to avoid sharing private training data. Yin et al. [13] tackle the vulnerability of FL to inference and poisoning attacks by applying encryption and blockchain-based reputation techniques to a FL framework for DDoS attack detection. In a recent paper, Popoola et al. [14] show the benefits of FL in detecting zero-day botnet attacks in Internet of Things (IoT) environments. The whole study is focused on the application of FEDAVG on traffic generated by infected IoT devices (including the Mirai [15] botnet) and compares FEDAVG against other training approaches, either centralised or distributed.

A common approach to tackle anomaly detection problems consists of training a Machine Learning (ML) model

with data collected during normal operations of the monitored environment (i.e., free from anomalies). However, in federated infrastructures, this might require sharing sensitive data among members of the federation. In this regard, FL has been exploited in recent works [16], [17], [18], [19], [20], [21], [22] to build privacy-preserving anomaly detection systems for IoT and computer networks. Although the proposed solutions show high detection accuracy scores, the use of the vanilla FEDAVG algorithm makes them prone to the drawbacks presented in Section 2. Finally, an interesting work by Wang et al. [23] presents a peer-to-peer variation of FL to train a model for anomaly detection in IoT without the need for a central server. To improve convergence and accuracy on non-i.i.d. and imbalanced data, clients share synthetic data with neighbours. Nevertheless, a stopping strategy for peer-to-peer training is not discussed.

In summary, we note that current threat detection solutions focus on performance (accuracy and communication rounds), with no or little attention to practical aspects. On the one hand, the assumption that test data is available at the central server location does not always hold. This is a common limitation of the works above, related to constraint C2 formulated in Section 2.2, in which it is not clear how the central server verifies the performance of the global model with respect to recent attacks. On the other hand, a few works rely on the vanilla FEDAVG algorithm [10], [11], [12], [14], [16], [20], [22], which aggregates the local models using weighted averaging. We will demonstrate in Section 8.1 that such an approach can greatly increase the convergence time on unbalanced non-i.i.d. attack data. Moreover, in some cases, constraint C1 is not respected (jeopardising clients' privacy), as data-sharing mechanisms are used to correlate non-i.i.d. features.

3.2 Unbalanced and non-i.i.d. data

The accuracy of ANNs trained with FEDAVG can degrade significantly in scenarios with class imbalance [24] or with non-i.i.d. data [25]. To mitigate the issues of unbalanced data across clients, Duan et al. [26] propose Astraea, a framework that combines data augmentation with mediators placed between the central server and clients. The role of each mediator is to reschedule the local training of a subset of clients, which are selected based on their data distribution. Zhang et al. [27] propose ranking the clients' models using their accuracy on a public test set before selecting the best-performing ones to be aggregated into a global IDS. Briggs et al. [28] apply a hierarchical clustering algorithm that uses clients' updates to determine the similarity of their training data. The algorithm returns a set of clusters, each containing a subset of clients with similar data. Wang et al. [24] propose a centralised monitoring system to spot class imbalance in the training data. The monitor relies on clients' data (part of it) to estimate the composition of data across classes. Zhao et al. [25] demonstrate the weight divergence of FEDAVG on non-i.i.d. data and improve the accuracy of the global model with a strategy that relies on sharing training data among clients. FAVOR [29] improves the performance of FEDAVG on non-i.i.d. data with a client selection mechanism based on reinforcement learning. An agent, collocated with the FL server, is in charge of selecting the clients that perform

computation at each round. The agent takes its decisions using a reward function that evaluates the accuracy of the global model on validation data.

We observe that none of the above approaches respects constraint C2, as all of them assume test data at the server location to assess model convergence. Moreover, the works of Wang and Zhao rely on sharing portions of clients' data, failing to meet constraint C1.

3.3 Efficient Federated Learning

The efficiency of the FL process is particularly relevant in the edge computing domain, where nodes possess a limited amount of resources (compared to cloud environments) to devote to critical or latency-sensitive tasks. In the scientific literature, the problem has been tackled from various angles: optimisation of the local training process, reduction of communication overhead, and minimisation of the number of local training rounds assigned to clients.

The approach of Ji et al. [30] is to progressively decrease the fraction of clients that perform local computation, while reducing the amount of transmitted data by means of a mechanism that masks part of the parameters of local models. Sparse Ternary Compression (STC) is a protocol proposed by Sattler et al. [31] to compress upstream and downstream communications between server and clients. Evaluation results show that STC converges faster than FEDAVG on non-i.i.d. data with lower communication overhead. The adaptive mechanism proposed by Wang et al. [32] and FedSens [33] focuses on improving the local training process. The former optimises the number of local gradient descent steps S taken by the clients (edge nodes) while minimising resource consumption (e.g., time, energy, etc.) and global loss function. Similar to FLAD, this approach relies on the performance (local loss function) of the global model on local datasets to control the number of local training steps S. However, formulation and evaluation of the proposed approach focus on application scenarios where the amount of training data is equally distributed across clients, with global loss and model computed using weighted averaging. FedSens implements an asynchronous FL framework, where each client can choose at which round to perform local computation. The goal is to find the best trade-off between classification accuracy and energy consumption (which is a function of the frequency of local and global updates).

Among these four works, only the mechanism of Wang et al. [32] satisfies both constraints C1 and C2. However, it has been designed for balanced settings, in which a common value of gradient descent steps across clients is sufficient to achieve the target objectives of accuracy and efficiency. We demonstrate in Section 8.1 that the weighted averaging adopted in that work prevents the global model to learn small and out-of-distribution attack classes in a reasonable training time. We also show that assigning specific training parameters to clients (based on the performance of the global model in their validation sets) greatly reduces convergence time.

Although FL offers the potential for collaborative training of deep learning models for DDoS detection, existing approaches are limited by their suitability for real-world

implementations, where new attack profiles must be shared with other partners with privacy guarantees. In this direction, we present our FL approach to DDoS detection FLAD, which respects constraints C1 and C2, while achieving high detection accuracy across all attack types in a reasonable training time. We demonstrate that FLAD is robust to model re-training upon the availability of new attack data. To the best of our knowledge, this is the first study in which the latter aspect is analysed and addressed.

4 THREAT MODEL

We consider a scenario in which the federation is composed of a set of clients that might belong to different organisations, plus an additional entity that manages the FL process (the central server). We assume that no one in the federation has the willingness/permission to share network traffic data with others. On the other hand, the federation's goal is to enhance the DDoS detection capabilities of each client' IDSs with attack profiles owned by other members.

In such a scenario, the clients are vulnerable to zero-day DDoS attacks at any given moment. To ensure the highest level of security, our system requires the global model to be updated promptly with the latest information, empowering all participants with the most recent detection features available. However, it is important to note that the central server may not always have access to network traffic profiles associated with these new and evolving threats. As a result, verifying the effectiveness of the global model in classifying such attacks becomes a challenge.

We also assume that neither the server nor the clients are malicious, thus they do not try to compromise the global model with poisoned data (e.g., weights obtained with mislabelled samples). Poisoning attacks can be a serious concern for IDSs that rely on collaborative training techniques for their operation. Malicious clients can manipulate the training process by providing mislabeled data or specially crafted samples, resulting in a final model that fails to accurately classify certain types of attack traffic. This problem is prevalent in most machine learning-based cybersecurity applications, including FL-trained DDoS attack detection systems. While poisoning attacks remain a critical concern for IDSs, previous research has already tackled the issue [13], [34], [35], [36], [37], and it is not within the scope of this work.

In this context, the adversary does not belong to the federation and does not have the knowledge to generate adversarial evasion attacks against the global ANN model [38]. However, it knows the IP addresses of the victims and how to generate DDoS attacks using spoofed network packets with the source IP address of the victims.

5 METHODOLOGY

FLAD enhances FEDAVG to solve the problem formulated in Section 2.2. In summary, the clients share with the server the classification score obtained by the global model on their local validation sets. As the server has a full view of the performance of the global model across the clients and their attacks, it can implement a training-stopping strategy that ensures acceptable performance on all the attack types, with

TABLE 1: Glossary of symbols.

w_t	Global model at round t
w_t^c	Model trained by client c at round t
$\bar{w}$	Trained global model
C	Set of federated clients
$c \in C$	A participant (client) in the FL process
c_e	Number of epochs assigned to client c
c_s	Number of MBGD steps/epoch assigned to client c
C_t	Subset of clients that perform training at round t
a^c	Accuracy score computed by client c on its validation set
a^μ	Average value of a^c computed over all $c \in C$
T_s^c	Time taken by client c to complete an MBGD step
T_n^c	Total time taken by the two-way transmission of the global model between server and client c
e_{min}, e_{max}	Minimum and maximum local training epochs
s_{min}, s_{max}	Minimum and maximum local training MBGD steps

no need for test data of all attacks (not always available at the server side, especially in the case of clients experiencing zero-day attacks). Additionally, the server uses this information to dynamically tune the computational workload of clients during each training round. This approach aims to accelerate the FL process when dealing with out-of-distribution (o.o.d.) data, or to alleviate the computational burden on clients whose local attack profiles are rapidly learnt.

With the term "computation", we refer to the number of training epochs/round (c_e) and the number of steps/epoch in Mini-Batch Gradient Descent (MBGD) (c_s). By multiplying these two values, we obtain the total number of MBGD steps/round allocated to a client for training the global model on its local dataset. In general, when training a neural network, the weights of the network can be thought of as a point in a high-dimensional space, where each dimension corresponds to an individual weight. The objective of the clients' local training process is to find the point that maximises the global model's accuracy on the local validation sets. In this regard, FLAD adopts a personalised training strategy that assigns a specific number of MBGD steps per round to each client. This allocation is based on the gap between the current global model's accuracy on the client's validation set and the maximum achievable accuracy. As a result, clients with larger accuracy gaps are required to perform more training steps to converge towards the optimal accuracy point. Conversely, clients with smaller gaps are assigned fewer steps, or even no steps at all, recognising their proximity to the point of maximum accuracy score.

Unlike FEDAVG, which assigns a fixed amount of computation (values of c_e and c_s) to a randomly selected subset of clients at every round, with this approach we aim to save clients' computing resources and to reduce the overall federated training time. This total training time is defined as the cumulative duration of all training rounds until convergence is achieved. In this regard, as the clients train in parallel, the time taken by a round of FL depends on the slowest client of the federation, as expressed in Equation 2.

$$T = \max_{c \in C_t}\{T_n^c + (c_e \cdot c_s) \cdot T_s^c\} \qquad (2)$$

The round time T is computed as the maximum training

time across the subset of clients C_t selected by FLAD at round t. The time spent by a client in a round of FL can be computed as the sum of the network time and computation time. The network time T_n^c is the time necessary for the two-way transmission of the global model between server and client. This time mostly depends on the type and the stability of the communication channel between the two parties. The computation time can be expressed as the sum of the time taken by all MBGD steps executed by the client during the FL round. The computation time is the result of the multiplication of the number of training epochs/round c_e by the number of MBGD steps/epoch c_s, by the time T_s^c taken by each step. This time depends on the size of the local training set of the client and the computational power of the client's hardware. Our intuition is that the convergence performance of FEDAVG can be improved by using the clients' classification scores to smartly select the clients at each round and to set per-client and per-round values of c_e and c_s. By reducing, or even eliminating, the computational workload assigned to clients whose traffic profiles are learned faster, we have the potential to optimise the overall convergence time.

We present the details of the federated training process executed by the server with FLAD in Algorithms 1 and 2, while the local training executed by clients is presented in Algorithm 3. The symbols are defined in Table 1.

The pseudo-code in Algorithm 1 describes the main process executed by the server, which orchestrates the operations of the clients. The algorithm takes as input a global model (w_0) and the set of clients involved in the FL process (C). It runs indefinitely until convergence is reached, as controlled by parameter PATIENCE, which is the number of rounds to continue before exit if no progress is made. The federated learning starts with the initialisation of the variables that are used to record the best global model along the process (max accuracy score a_{max}) and to implement the early stopping strategy (counter sc keeps track of the rounds with no improvements in average accuracy score a^μ). At line 5, the amount of computation for the clients is set to the maximum values of training epochs and MBGD steps. The loop at lines 8-10 triggers the CLIENTUPDATE methods (Algorithm 3) for a subset of selected clients C_{t-1}. Note that at round $t = 1$, $C_{t-1} = C_0 = C$, i.e., the input set of clients (line 4).

At each round, the server computes the average of the parameters from all clients, regardless of whether they were involved in the previous round of training (line 11). Please note that to speed up convergence on unbalanced and non-i.i.d. data across clients, FLAD replaces the weighted mean in Equation 1 with the arithmetic mean, similarly to other works in literature (e.g., [31], [5], [7]). The new global model is sent to all clients, which return the accuracy scores $[a^c]_{c \in C}$ obtained on their local validation sets with the new global model (line 12). The server computes the mean accuracy score value a^μ, which is used to evaluate the progress of the federated training (lines 13-19). If $a^\mu > a_{max}$, the new global model is saved and the stopping counter sc is set to 0. Otherwise, sc is increased by one to record no improvements. When $sc > $ PATIENCE (in our experiments we set PATIENCE $= 25$ rounds), the process stops and the best model is sent to all the clients for integration in their

---

Algorithm 1 Adaptive federated learning process.

Input: Global model (w_0), set of clients (C)
1: **procedure** ADAPTIVEFEDERATEDTRAINING
2:     $a_{max} \leftarrow 0$ ▷ Max accuracy score
3:     $sc \leftarrow 0$ ▷ Early stop counter
4:     $C_0 \leftarrow C$
5:     $c_e = e_{max}, c_s = s_{max} \; \forall c \in C_0$ ▷ Epochs and steps
6:     $c \leftarrow$ INITCLIENTS(w_0, c_e, c_s) $\forall c \in C_0$
7:     **for** round $t = 1, 2, 3, ...$ **do** ▷ Federated training loop
8:         **for all** $c \in C_{t-1}$ **do**   ▷ In parallel
9:             $w_t^c \leftarrow$ CLIENTUPDATE(w_{t-1}, c_e, c_s)
10:         **end for**
11:         $w_t = \frac{1}{|C|} \sum_{c=1}^{|C|} w_t^c$ ▷ Arithmetic mean
12:         $a^\mu \leftarrow [a^c]_{c \in C} \leftarrow$ SENDMODEL(w_t, C)
13:         **if** $a^\mu > a_{max}$ **then**
14:             $\bar{w} \leftarrow w_t$ ▷ Save best model
15:             $a_{max} \leftarrow a^\mu$ ▷ Save max accuracy score
16:             $sc \leftarrow 0$ ▷ Reset early stop counter
17:         **else**
18:             $sc \leftarrow sc + 1$
19:         **end if**
20:         **if** $sc > $ PATIENCE **then**
21:             SENDMODEL($\bar{w}, C$)   ▷ Send final model
22:             **return**   ▷ End of the process
23:         **else**
24:             $C_t \leftarrow$ SELECTCLIENTS($C, [a^c]_{c \in C}, a^\mu$)
25:         **end if**
26:     **end for**
27: **end procedure**

IDSs (line 21). Otherwise, the server calls Algorithm 2 to determine which clients will participate in the next round and to assign the number of epochs and MBGD steps to each of them.

Algorithm 2 Select the clients for the next round of training.

Input: Clients (C), accuracy scores ($[a^c]_{c \in C}$), average accuracy score (a^μ)
Output: List of selected clients (C')
1: **procedure** SELECTCLIENTS($C, [a^c]_{c \in C}, a^\mu$)
2:     $C' \leftarrow \{c \in C \mid a^c \leq a^\mu\}$
3:     $\underline{a} = min_{c \in C'}(a^c)$
4:     $\overline{a} = max_{c \in C'}(a^c)$
5:     **for all** $c \in C'$ **do**
6:         $\sigma = \frac{\overline{a} - a^c}{\overline{a} - \underline{a}}$ ▷ Scaling factor
7:         $c_e = e_{min} + (e_{max} - e_{min}) \cdot \sigma$
8:         $c_s = s_{min} + (s_{max} - s_{min}) \cdot \sigma$
9:     **end for**
10:     **return** C'
11: **end procedure**

Algorithm 2 starts with selecting the subset of clients C' that will execute the local training in the next round. C' is the set of $c \in C$ whose accuracy score a^c obtained on their local validation set is lower than the mean value a^μ (line 2). The number of epochs and steps assigned to each client $c \in C'$ depends on the value of a^c. The rationale is that the higher a^c, the lower the amount of

computation needed from the client (thus, fewer epochs and MBGD steps/epoch, as explained at the beginning of this section). This is formalised in the equations within the loop at lines 5-9, where each client $c \in C'$ is assigned a minimum number of epochs/steps plus an additional amount that is inversely proportional to the accuracy score a^c. The scale factor σ ranges over $[0, 1]$, assuming value 0 when $a^c = max_{c \in C'}(a^c)$ (hence $c_e = e_{min}$ and $c_s = s_{min}$) and value 1 when $a^c = min_{c \in C'}(a^c)$ (hence $c_e = e_{max}$ and $c_s = s_{max}$). Algorithm 2 returns the set of clients C' that will perform computation during the next round, each assigned with a specific number of epochs and MBGD steps.

Algorithm 3 Local training procedure at client c.

Input: Global parameters w, epochs (c_e), MBGD steps (c_s)
Output: Updated parameters (w)
1: **procedure** CLIENTUPDATE(w, c_e, c_s)
2:     $X, y \leftarrow$ LOADDATASET()
3:     **if** $c_s > 0$ **then**
4:         $c_b \leftarrow max(|X_{train}|/c_s, 1)$ ▷ Compute batch size
5:     **end if**
6:     $\mathcal{B} \leftarrow$ split X_{train} into batches of size c_b
7:     **for** epoch e from 1 to c_e **do**
8:         **for all** batch $b \in \mathcal{B}$ **do**
9:             $w \leftarrow w - \eta \nabla L(w, b)$
10:        **end for**
11:    **end for**
12:    **return** w ▷ Return updated parameters to server
13: **end procedure**

The pseudo-code provided in Algorithm 3 outlines the local training procedure carried out by clients. This process starts from the weights and biases of the current global model w received from the server, and is executed for a number of epochs c_e and MBGD steps c_s assigned by the server. The first operation is the computation of the batch size c_b using c_s (line 4). It ensures that $c_b \geq 1$, for the cases in which the number of samples in the local training set is smaller than c_s. Once the batch size is computed, the algorithm continues with $c_e \cdot c_b$ steps of gradient descend (lines 7-11) and finally returns the updated model to the server.

6 THE DATASET

FLAD is validated with a recent dataset of DDoS attacks, CIC-DDoS2019 [39], provided by the Canadian Institute of Cybersecurity of the University of New Brunswick. CIC-DDoS2019 consists of several days of network activity, and includes both benign traffic and 13 different types of DDoS attacks. The dataset is publicly available in the form of pre-recorded traffic traces, including full packet payloads, plus supplementary text files containing labels and statistical details for each traffic flow [40]. The benign traffic of the dataset has been generated using the B-profile introduced in [41], which defines distribution models for web (HTTP/S), remote shell (SSH), file transfer (FTP) and email (SMTP) applications. Instead, the attack traffic has been generated using third-party tools and can be broadly classified into two main categories: *reflection-based* and *exploitation-based* attacks. The first category includes those attacks, usually

based on the UDP transport protocol, in which the attacker elicits responses from a remote server (e.g., a DNS resolver) towards the spoofed IP address of the victim. Hence, the victim is ultimately overwhelmed by the server's replies. The second category relates to those attacks that exploit known weaknesses of some network protocols (e.g., the three-way handshake of TCP). An overview of the CIC-DDoS2019 dataset is provided in Table 2.

In Table 2, the column *#Flows* indicates the amount of bi-directional TCP sessions or UDP streams contained in the traffic traces provided with the dataset, each flow identified by a 5-tuple (source IP address, source TCP/UDP port, destination IP address, destination TCP/UDP port and IP protocol). Before experimenting with our solution, we have pre-processed the traffic traces with the tool developed in our previous work LUCID [42]. The resulting representations of traffic flows are in the form of arrays of shape $n = 10$ rows and $f = 11$ columns. Each row contains a representation of a packet based on 11 features, the same considered in the LUCID paper: *Time, Packet Length, Highest Protocol, IP Flags, Protocols, TCP Length, TCP Ack, TCP Flags, TCP Window Size, UDP Length* and *ICMP Type*. If the number of packets of a flow is lower than n, the array is zero-padded. The number of non-zero rows in the array can be seen as another feature that we call *FlowLength*. It is worth recalling that packets are inserted into the array in chronological order and that the timestamp is the inter-arrival time between a packet and the first packet in the array. As the packet attributes are extracted using TShark [43], we can use some high-level features such as the highest protocol detected in the packet and the list of all protocols recognised in the packet. The LUCID dataset parser splits each traffic flow into smaller subsets of packets to produce samples that are consistent with real-world settings, where the detection algorithms must cope with fragments of flows collected over pre-defined time windows. Shorter time windows allow faster decisions, but also a higher fragmentation of the flows, hence a possible decrease in the classification accuracy. In this work, we use a time window of duration 10 seconds for both benign and attack traffic. By taking this choice, we slightly increase the size of the smallest attack (202 WebDDoS samples obtained by splitting 146 flows), while maintaining an adequate level of accuracy, as per evaluation results reported in the LUCID paper.

6.1 Feature distribution analysis

In the use-case scenario considered in this work, individual clients contribute to the federated training with private data collections of benign and attack traffic, possibly drawn from non-identical feature distributions. To compare the feature distributions among the attack types of the CIC-DDoS2019 dataset, we use the JSD [44] metric. JSD measures the degree of overlapping of two probability distributions, where distance zero means identical distributions, while distance one means that the two distributions are supported on non-overlapping domains. Figure 1 reports the JSD values for all features and attack types. More precisely, an element (i, j) in the matrix is the average JSD value between the attack at row i and each of the other attacks, computed on their probability distributions of the feature at column j.
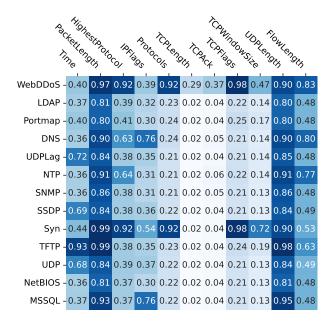
Fig. 1: Jensen-Shannon Distance (JSD) map of the probability distributions of the features.

Fig. 2: Probability density functions of the *Packet Length* feature.

In Figure 1, we observe that every attack presents at least one feature whose probability distribution domain is almost disjoint from those of the other attacks. As also shown in Figure 2, this primarily relates to features *Packet Length* and *UDP Length* (redundant in this dataset, where most of the attacks are UDP-based). Indeed, similar distributions with
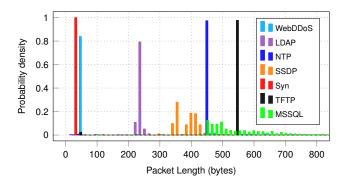
different packets sizes can be observed on *LDAP*, *NTP* and *TFTP* attacks, while the packet sizes in other attacks are distributed across larger domains (Figure 2). About TCP-based attacks, all the *Syn Flood* packets have *Packet Length* equal to 40 bytes, while more than 80% of the *WebDDoS* packets have a size of either 66 or 74 bytes.

Similar considerations apply to other features such as *Flow Length*, indicating that also the distribution of packets/sample changes across the different attacks. Finally, it is worth noting the large JSD distance of the two TCP-based attacks (WebDDoS and Syn Flood) from the other attacks (UDP-based) on most of the features. We will show in Section 8 the negative impact of such out-of-distribution attacks on the convergence of FEDAVG.

TABLE 2: Overview of the CIC-DDoS2019 DDoS attack types.

Attack	#Flows	Transport	Description
DNS **LDAP** **MSSQL** **NTP** **NetBIOS** **Portmap**	441931 11499 9559537 1194836 7553086 186449	UDP	DDoS attacks that exploit a specific UDP-based network service to overwhelm the victim with responses to queries sent by the attacks to a server using the spoofed victim's IP address. Six types of network services have been exploited to generate these attacks: Domain Name System (DNS), Lightweight Directory Access Protocol (LDAP), Microsoft SQL (MSSQL), Network Time Protocol (NTP), Network Basic Input/Output System (NetBIOS) and Port Mapper (Portmap).
SNMP	1334534	UDP	Reflected amplification attack leveraging the Simple Network Management Protocol (SNMP) protocol (UDP-based) used to configure network devices.
SSDP	2580154	UDP	Attack based on the Simple Service Discovery Protocol (SSDP) protocol that enables UPnP devices to send and receive information over UDP. Vulnerable devices send UPnP replies to the spoofed IP address of the victim.
TFTP	6503575	UDP	Attack built by reflecting the files requested to a Trivial File Transfer Protocol (TFTP) server toward the victim's spoofed IP address.
Syn Flood	6056402	TCP	Attack that exploits the TCP three-handshake mechanism to consume the victim's resources with a flood of SYN packets.
UDP Flood	6969476	UDP	Attack built with high rates of small spoofed UDP packets with the aim to consume the victim's network resources.
UDPLag	474018	UDP	UDP traffic generated to slow down the victim's connection to the online gaming server.
WebDDoS	146	TCP	A short DDoS attack (around 3100 packets) against a web server on port 80.
Total	42865789		Despite the huge amount of flows, the dataset is heavily imbalanced, containing 8 predominant DDoS attack types, with more than one million flows each, a few tenths of thousands flows for the LDAP and Portmap reflection attacks, and only 146 flows for the WebDDoS attack.

7 EXPERIMENTAL SETUP

The FLAD approach is validated using a fully connected neural network model (or MLP), which is initialised with random parameters (weights and biases) by the server and locally trained multiple times by the clients, as per the procedure presented in Section 5.

As we are interested in measuring the benefits of FLAD over other approaches (FEDAVG and FLDDoS [5]) in terms of load on the clients, convergence time and classification accuracy, we want to avoid the impact of communication inefficiencies, such as network latencies that can occur in distributed deployments. Therefore, FLAD is implemented as a single Python process using Tensorflow 2.7.1 [45], thus server and clients are executed on the same machine and communicate through local procedure calls. Please note that this implementation choice does not affect the validity or generality of our work. Federated training and model testing have been performed on a server-class computer equipped with two 16-core Intel Xeon Silver 4110 @2.1 GHz CPUs and 64 GB of RAM.

7.1 Dataset preparation

The CIC-DDoS2019 dataset has been split into 13 smaller datasets, each containing samples of benign traffic and only one type of attack. Furthermore, we deliberately introduced an imbalance across the 13 datasets by doubling the number of samples from one dataset to another, starting from the one with the smallest attack (202 WebDDoS samples) and culminating in the largest dataset (MSSQL), which has been reduced to 819204 attack samples. Every dataset split has been carefully balanced to ensure an approximately equal distribution between benign and DDoS samples. The balanced datasets were further divided into training (90%) and test (10%) sets, with an additional 10% of the training set reserved for validation purposes. (Table 3).

TABLE 3: CIC-DDoS2019 dataset splits.

Dataset split	Samples	Training	Validation	Test
WebDDoS	402	321	37	44
LDAP	854	633	135	86
Portmap	1605	1299	145	161
DNS	3207	2595	291	321
UDPLag	6400	5184	576	640
NTP	12807	10372	1153	1282
SNMP	25649	20775	2309	2565
SSDP	51207	41477	4609	5121
Syn Flood	102400	82940	9216	10244
TFTP	204800	165887	18433	20480
UDP Flood	409601	331772	36864	40965
NetBIOS	819200	663551	73728	81921
MSSQL	1638404	1327105	147457	163842

The dataset splits outlined in Table 3 serve as an evaluation framework for FLAD under a worst-case scenario. In this scenario, each attack is exclusively assigned to a single client (one-to-one mapping), resulting in a pathological non-i.i.d. partition of the data, as referred to by McMahan et al. [4]. Furthermore, these splits can be combined to create larger federations to assess scalability or to replicate experimental settings employed by other state-of-the-art approaches for comparison purposes.

7.2 ANN architecture

The architecture of our MLP model consists of an input layer of shape $n \times f$ neurons, a single-neuron output layer and l hidden dense layers of m neurons each (Figure 3). The input of the neural network is an array-like representation of a traffic flow, where lines are packets of the flow in chronological order from top to bottom, and columns are packet-level attributes (see Section 6). Before processing, each array is reshaped into a $n \cdot f$-size vector, where packets are lined up one after another in chronological order. The objective of the
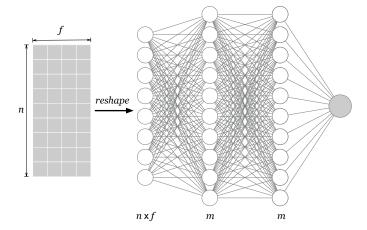

Fig. 3: Architecture of the ANN used to evaluate FLAD.

local training procedure, summarised in Algorithm 3, is to minimise the cross-binary cost function defined in Equation 3. The cost function measures the quality of the model's traffic classification compared to the ground truth of the input. At each training epoch, the error is back-propagated through the network and it is used to iteratively update the model's weights until convergence is obtained.

$$c = -\frac{1}{s} \sum_{j=1}^{s} (y_j \log p_j + (1 - y_j) \log(1 - p_j)) \quad (3)$$

In Equation 3, y_j is the ground truth label of each input flow j in a batch of s samples. The label of benign flows is equal to 0, while the label of DDoS flows is equal to 1. The value of $p_j \in (0, 1)$ is the predicted probability flow j is DDoS. The cost c, as computed in Equation 3, tends to 0 when the output probabilities of the flows are close to the respective ground truth labels.

Note that, the cost is computed by each client independently, using only local training data. In the case of non-i.i.d. features across clients (cf. Section 6.1), the distance between the data distributions can lead the weights of different clients to diverge, slowing down the convergence of the FL process to the target performance of the global model [25].

7.3 FL hyper-parameters

The whole FL process is configured with a set of hyper-parameters, which have been determined either based on the results of the preliminary tuning activities (PATIENCE, MLP architecture), or based on the observations of McMahan et al. [4] on local training epochs and batch size. The hyper-parameters presented in Table 4 have been used to

validate FLAD and to compare it against FEDAVG and Federated Learning DDoS (FLDDoS) [5], a recent FEDAVG-based solution for DDoS attack detection introduced in Section 3 and presented below. The values in the table have been kept constant across all experiments described in Section 8.

TABLE 4: Hyper-parameters of FLAD.

Name	Value	Description
PATIENCE	25	Max FL rounds with no progress.
Min epochs	1	Min number of local training epochs.
Max epochs	5	Max number of local training epochs.
Min steps	10	Min number MBGD steps.
Max steps	1000	Max number MBGD steps.
$n \times f$	10×11	Size of the MLP input layer.
l	2	Number of hidden layers.
m	32	Number of neurons/layer.

The value of PATIENCE has been set to 25 rounds, which, compared to lower values, guarantees good accuracy on small and non-i.i.d. attacks, such as WebDDoS and Syn Flood. In terms of the number of hidden layers and activations, we started with larger architectures and then progressively reduced the dimensions until we reached a configuration that allowed good detection accuracy on all attacks in a reasonable time. The dynamic tuning of epochs and MBGD steps implemented by FLAD is configured with the ranges presented in Table 4. The minimum and maximum values of epochs have been set as the same values used to evaluate FEDAVG. Unlike other approaches such as FEDAVG and FLDDoS, we do not specify the batch size, but instead, we tune the number of MBGD steps each client has to take at each round, hence controlling the client's training process without having to consider the size of its local dataset. We experiment with amounts of steps ranging between 10 and 1000.

Concerning our implementation of FEDAVG and FLDDoS, we use the same MLP architecture used for testing FLAD, with the same values of hyper-parameters n, f, l and m reported in Table 4. The remaining hyperparameters, such as the number of local epochs E and batch size B (described in Section 2), are set according to the values specified in the respective papers [4], [5].

In the case of FEDAVG, we experiment with $E = 1$ and $E = 5$ epochs/round of local training and with a fixed batch size of $B = 50$ samples. Other values could be also chosen (e.g., $E = 20$ or $B = 10$, also used by McMahan et al. in their work), but after a preliminary investigation, we found that the little gain in accuracy achieved with further MBGD steps was not sufficient to balance the impressive amount of additional local computation on clients with large datasets. In their study, McMahan et al. assess the performance of FEDAVG using a client fraction $F = 0.1$. For their evaluation, they form federations consisting of 100, 600, and 1000+ clients, focusing on tasks such as image classification, digit recognition, and language modelling. However, in our experiments, we primarily work with smaller federations ranging from 13 to 90 clients. Therefore, we align with FLDDoS authors' recommendation for testing FL in a DDoS attack detection scenario and we set F to 0.8.

FLDDoS aims to mitigate the limitations of FEDAVG on non-i.i.d. DDoS attack data by maintaining a local model at each client. More precisely, at round t, each client $c \in C$ updates the global model w_t^c as done with FEDAVG. In addition, the client maintains a local model v_t^c that is trained solely with the local data. These two models are then merged to create a *personalised model* $\bar{v}_t^c$, which is subsequently sent to the server for aggregation. The formula used to calculate the personalised model is as follows:

$$\bar{v}_t^c = \gamma^c w_t^c + (1 - \gamma^c) v_t^c \quad \forall c \in C \qquad (4)$$

Where the weighting factors γ^c can be tuned based on the level of heterogeneity in the clients' data. Note that when $\gamma^c = 1 \ \forall c \in C$, FLDDoS corresponds to FEDAVG. Otherwise, by setting $0 \le \gamma^c < 1$ for $c \in C$, one can allow a client to tune the relative importance of the local model v_t^c in the personalised model that is sent to the server for aggregation. Although the authors of FLDDoS do not suggest any possible values for γ^c, in our experiments we try to understand the impact of model personalization on the convergence of the FL process. To this purpose, we test FLDDoS with $\gamma^c = 1$ (no local model as for FEDAVG) for those clients contributing to the FL with UDP-based attacks, and $\gamma^c = 0.9$ for the clients with TCP-based attacks, namely WebDDoS and SYN Flood. As demonstrated in Section 8.1.1, learning such attacks can be particularly challenging using the FEDAVG since these are the only two TCP-based attacks in the dataset, while the majority of attacks are UDP-based. By setting $\gamma^c = 0.9$, we try to increase the contribution of the TCP-based attacks when building the global model and to understand whether the FLDDoS approach addresses the limitations of FEDAVG.

Additionally, we use the hyper-parameter values reported in the paper, including a fraction of clients $F = 0.8$, a batch size of $B = 100$ samples, and $E = 10$ local epochs per round.

8 EXPERIMENTAL EVALUATION

The evaluation focuses on assessing the adaptive approach of FLAD in terms of convergence time of the FL process, DDoS attack detection accuracy of the global model, and scalability. For this purpose, we utilize the CIC-DDoS2019 dataset, which is configured according to the specifications outlined in Section 7.1.

The classification performance of FLAD is measured in terms of *F1 score* and *True Positive Rate (TPR)*. The TPR, also called *Recall*, is the ratio between the correctly detected DDoS samples and all the DDoS samples in the dataset. The TPR quantifies how well a model can identify the DDoS attacks.

The *F1 Score* is a widely used metric to evaluate classifier accuracy, computed as the harmonic mean of *Precision* and TPR, with *Precision* (Pr) being the ratio between the correctly detected DDoS samples and all the detected DDoS samples. The *F1 Score* is formally defined as $F1 = 2 \cdot \frac{Pr \cdot TPR}{Pr + TPR}$. The *F1 Score* is also used as the accuracy metric in the implementation of Algorithms 1 and 2 presented in Section 5.

8.1 State-of-the-art comparison

We compare FLAD against FEDAVG, the original FL algorithm proposed by McMahan et al. [4] and against a recent FL-based solution for DDoS attack detection called FLDDoS [5]. Both FEDAVG and FLDDOS adopt a randomised client selection strategy, while also employing fixed batch sizes and local training epochs across all clients. The goal of this evaluation is to expose the limitations of such design choices in a cybersecurity scenario, where the server does not possess a test set (for the reasons discussed earlier in this paper) to measure the performance of the global model on different attack types.

8.1.1 Convergence analysis

In this experiment we train the global model with FLAD until convergence, i.e., waiting for PATIENCE=25 rounds with no progress in the average F1 Score across the clients. Following this, we evaluate the performance of the original FEDAVG algorithm and FLDDoS by subjecting them to the same number of training rounds as FLAD.

We perform the convergence analysis in a worst-case scenario, i.e., with a federation of 13 clients and a one-to-one mapping between clients and DDoS attack types. We replicate the same experiment by employing a federation of 50 clients, each containing two attack types in their local dataset. The latter settings align with Lv et al.'s evaluation of FLDDoS in their study [5].

Each experiment is repeated 10 times and the average metrics are reported in this section. As TensorFlow relies on a pseudo-random number generator to initialise the global model, and both FEDAVG and FLDDoS perform a random selection of clients at each FL round, each experiment is initiated with a unique random seed to ensure diverse testing conditions.

TABLE 5: Average metrics over 10 experiments in the 13-client scenario, with one-to-one clients/attacks mapping.

Metric	FLAD E=A,S=A	FedAvg E=1,B=50	FedAvg E=5,B=50	FLDDoS E=10,B=100
FL Rounds	68	68	68	68
Round Time (sec)	9.08	34.19	179.48	205.39
Total Time (sec)	617	2325	12205	13967
F1 Score	0.9667	0.8577	0.9157	0.9091
F1 StdDev	0.0369	0.2714	0.1597	0.1605
F1 WebDDoS	0.8990	0.0815	0.8148	0.7376
F1 Syn	0.9877	0.4563	0.4613	0.5094

The results obtained in the worst-case scenario are summarised in Table 5, which reports average metrics across the 10 iterations of this experiment. As introduced in Section 7.3, FLAD is configured with adaptive tuning of epochs and MBGD steps of local training (E=A,S=A). FLAD is compared against two configurations of FEDAVG, with E=1 and E=5 epochs/round of local training, and against FLDDoS configured with E=10.

The table shows the advantages of FLAD over FEDAVG and FLDDoS: higher accuracy within a shorter time frame. These improvements can be attributed to the dynamic client selection strategy implemented by FLAD. At each round of the federated training process, clients are chosen based on the performance of the current aggregated model on their local datasets. Consequently, FLAD prioritizes attacks that are more challenging to learn, specifically the o.o.d. attacks WebDDoS and Syn Flood. The clients with these attacks are selected more frequently for local training, with an average of approximately 44 and 46 rounds respectively out of a total of 68 rounds, compared to an average of around 18 rounds for the clients with the other attacks.

In contrast, both FEDAVG and FLDDoS rely on random client selection, where each client is involved in approximately 77% of the training rounds (around 52 rounds on average out of a total of 68 rounds), considering the client fraction $F = 0.8$ used in our experiments. This results in longer rounds due to the frequent inclusion of clients with large local datasets, even when their contribution is not essential for improving the accuracy of the aggregated model. Furthermore, FLAD dynamically tunes the amount of computation assigned to the selected clients at each round of training, resulting in a significant reduction in the average local training time per round. Comparatively, FLAD achieves an average local training time of around 9 seconds per round, while the two configurations of FEDAVG require 34 and 179 seconds per round, respectively. The FLDDoS configuration, on the other hand, takes more than 200 seconds per round. Consequently, FLAD's adaptive allocation strategy not only decreases the per-round training time but also effectively reduces the overall duration of the federated training process.

It is also worth noting that the overall performance of FLDDoS and FEDAVG with E=5 is similar, as they assign approximately the same amount of computation to the clients. Specifically, FLDDoS is configured with E=10 epochs local training (as in the original paper by Lv et al. [5]), while FEDAVG uses E=5 epochs with twice the number of MBGD steps/epochs due to the smaller batch size. Additionally, the strategy employed by FLDDoS to handle non-i.i.d. data does not yield significant improvements compared to FEDAVG in our evaluation scenario. In fact, the local models maintained by clients with o.o.d. data, such as WebDDoS and Syn Flood attack traffic, do not contribute to improving the accuracy of the global model on such attacks but, instead, increase the total training time.

This is clearly shown in Figure 4, which shows the performance trend of FLAD, FEDAVG and FLDDoS on the WebDDoS and Syn Flood attack traffic during the first of the 10 iterations of the experiment. The two plots clearly demonstrate that FLAD achieves faster learning and higher accuracy for both of these attacks, while FLDDoS and FEDAVG with E=5 exhibit similar trends.

In these plots, we can also observe the adaptive mechanism of FLAD in action. Once the global model has learnt a client's data profile, FLAD excludes such client from the next round of federated training. In the case of clients with o.o.d. data, such as WebDDoS and Syn attack data, this behaviour might cause the model to forget what it has previously learnt on such attacks, as can be seen on both plots in the figure. However, this prompts FLAD to reintegrate such clients in the subsequent rounds of the training process, ultimately leading to global convergence.

Finally, Figure 5 presents the performance trend of FLAD, FEDAVG, and FLDDoS in a scenario with 50 clients, where each client's dataset consists of two attacks along
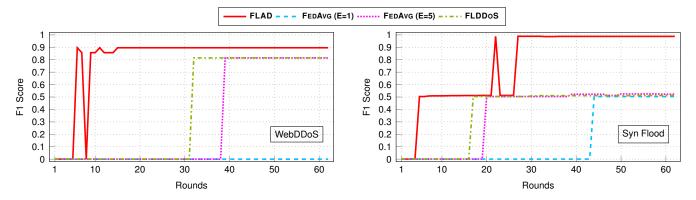
Fig. 4: Performance on out-of-distribution data, namely on the WebDDoS and Syn Flood attacks.

with benign traffic. Also in this case, we repeated the experiment 10 times. However, due to limited space, only the test results of the first iteration are displayed. Nevertheless, a similar pattern was observed throughout the remaining nine iterations. The local datasets of the 50 clients are generated by randomly combining pairs of the 13 datasets listed in Table 3. For each test iteration, a different random seed is utilised to generate a distinct federation.
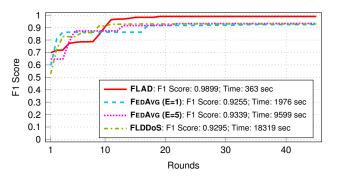

Fig. 5: Performance comparison on a federation of 50 clients, two attacks/client.

It is worth mentioning that, given these test settings, the two o.o.d. attacks are present in the datasets of multiple clients. Due to this, we observe a higher average F1 score on the clients' validation sets compared to the 13-client scenario (approximately 0.99 with FLAD and 0.94 with FLDDoS and the two configurations of FEDAVG) and lower standard deviation across the 50 local validation sets (approximately 0.01 with FLAD and 0.1 with FLDDoS and FEDAVG). These results demonstrate the advantages of the adaptive mechanism implemented by FLAD, even in scenarios with more uniformly distributed and less imbalanced data.

8.1.2 Evaluation on unseen data

To assess the performance of the global models trained using FLAD, FEDAVG, and FLDDoS, we conduct evaluations on previously unseen data. To this purpose, we use the models trained in the worst-case scenario of 13 clients (Section 8.1.1). Thus, we test the capability of the global models to correctly recognise the 13 attacks. To this aim, Table 6 reports the average TPR measured on the test sets of the clients using the final models obtained in the 10 experiments. While FLAD produces high TPR values

across all the attacks, the results of this experiment highlight the shortcomings of FEDAVG and FLDDoS when dealing with out-of-distribution attack traffic (the TCP-based attacks WebDDoS and Syn Flood). These findings validate the conclusions drawn from the aggregated metrics analysis presented in the previous section.

TABLE 6: Comparison on clients' test sets (TPR).

Attack	FLAD E=A,S=A	FedAvg E=1,B=50	FedAvg E=5,B=50	FLDDoS E=10,B=100
WebDDoS	0.7864	**0.0727**	**0.7182**	**0.6455**
LDAP	0.9306	0.8972	0.9306	0.9083
Portmap	0.9250	0.8548	0.9221	0.8740
DNS	0.9779	0.9060	0.8799	0.8772
UDPLag	0.9652	0.9978	0.9984	0.9981
NTP	0.9660	0.9874	0.9701	0.9661
SNMP	0.9574	0.9211	0.9586	0.9320
SSDP	0.9663	0.9983	0.9988	0.9989
Syn	0.9767	**0.3188**	**0.3254**	**0.3861**
TFTP	0.9439	0.9483	0.9372	0.9524
UDP	0.9656	0.9996	0.9995	0.9995
NetBIOS	0.9218	0.8581	0.9272	0.8820
MSSQL	0.9981	0.9994	0.9176	0.9503
Average	0.9699	0.9396	0.9155	0.9232

8.1.3 Discussion

The key improvement of FLAD over to FEDAVG (and other solutions based on it, such as FLDDoS) is the mechanism that monitors the performance of the global model, which allows the implementation of adaptive methods to control the FL process and the definition of a stopping strategy. About the latter, in our experiments, we stop the FL process after PATIENCE=25 rounds with no progress in the average F1 score. Alternatively, more advanced stopping strategies are also possible with FLAD. For instance, the practitioner might want to wait longer until a target average accuracy is reached, perhaps also combined with a target standard deviation of the accuracy scores to ensure that the performance is stable across all local datasets.

8.2 Federated re-training with new attack data

We now evaluate FLAD in a realistic scenario, where the global model needs frequent retraining to learn new attacks.

This experiment starts with two clients training on attack data (one attack type each) and benign traffic (Algorithm 1

where w_0 is a set of randomly initialised parameters and $|C| = 2$). Once the federated training process converges, the resulting aggregated model is used as a starting point for another round of federated training (Algorithm 1 with $w_0 = \bar{w}$), in which we provide attack data (a new attack type) to a third client ($|C| = 3$) to simulate the discovery of a new zero-day DDoS attack. Once convergence is achieved, we restart training by introducing new attack data on a fourth client. This is repeated until all thirteen attacks have been added, one on each client, and all the clients are provided with a model that has been trained with all the attack profiles. Each retraining iteration should converge as soon as possible and should produce aggregated models with high classification scores across the available attacks (high average F1 score with low standard deviation).

At each step of this experiment, we stop the FL process after PATIENCE=25 rounds with no progress in the average F1 score, and we start again by introducing a new attack as described above. As results may depend on the order in which attacks are introduced into the experiment, we repeat the whole experiment 10 times, each time with a different sequence of attacks.
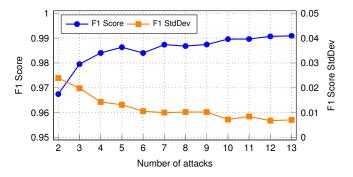

Fig. 6: Mean and standard deviation of the F1 score at an increasing number of DDoS attacks.

The experiment's findings have been visually presented in Figure 6. The plot in the Figure captures the progression of the average F1 score and its corresponding standard deviation throughout the various stages of the experiment. The *F1 Score* is the average F1 Score measured on the validations sets reported by the clients each time convergence is achieved at each step of the experiment (hence with 2, 3, ..., 13 attacks) averaged over the 10 experiments). Similarly to the F1 score, we compute the *F1 StdDev* as the average standard deviation of the F1 Scores on the validation sets of the clients for each step of the experiment.

As we can observe in the figure, FLAD produces high performance across various attack combinations. Regardless of the attack types used, FLAD consistently achieves an F1 score above 0.96, with a remarkably low standard deviation of less than 0.025. The results obtained demonstrate the adaptability of FLAD in adjusting its learning strategy in response to changes in data distribution and imbalance ratio. This adaptability extends to various scenarios, including those where only a few attacks are present, as well as instances involving out-of-distribution features, such as the two TCP-based attacks.

However, in scenarios where only two attacks are present, we observe a slight drop in performance. This can

be attributed to the adaptive selection of clients, which sometimes allocates computation to one client while leaving the other with none. Particularly when the feature distribution of the two attacks is significantly different, this dynamic assignment of computation may cause the F1 score of the global model to fluctuate between the two attack types, without any notable improvement in the average F1 score. Consequently, there are instances where the FL process halts prematurely due to the "patience" mechanism, before achieving the expected average accuracy. In corner cases like this, it may be necessary to implement more sophisticated stopping strategies, as previously mentioned in Section 8.1.3.

8.3 Scalability analysis

The previous sections detail experiments conducted in scenarios with federations of either 13 or 50 clients. However, these experiments do not provide insight into FLAD's stability when handling larger federations. To address this, we assess FLAD's performance across a range of increasing federation sizes, from 13 to 90 clients, measuring key metrics such as F1 Score and time required to reach convergence (inclusive of the 25 rounds of patience).

To create federations of increasing sizes, we have progressively augmented the original set of 13 clients with new clients generated by selecting two local datasets from the original set in various combinations. This process can produce up to 78 new clients, each with a distinct local dataset, as calculated by the formula $n!/(k!(n-k)!)$ with $n = 13$ and $k = 2$, for a maximum federation size of $13 + 78 = 91$ clients.

The experiment has been executed ten times for each federation size, with random subsets of clients selected at each iteration. The resulting average trends of F1 Score and convergence time are displayed in Figure 7.
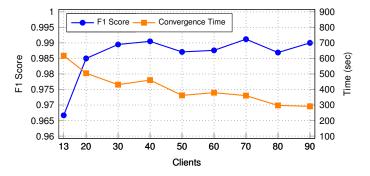

Fig. 7: F1 Score and convergence time as functions of federation size.

The Figure clearly illustrates that FLAD exhibits consistent stability in performance as the number of clients increases. Furthermore, these results validate our initial assumption that the one-to-one mapping between clients and attack types presents the most challenging scenario for the convergence of the FL process. The data reveals the lowest F1 Score and the longest average convergence time during our experiments, supporting this conclusion.

Please keep in mind that the F1 Score for the global model is calculated by averaging the F1 Scores on all clients'

validation sets. Therefore, when the number of clients is small, a low score on a few validation sets can have a significant impact on the overall average. In the scenario with 13 clients, we observe an average F1 Score ranging between 0.90 and 0.97 on the validation sets of seven clients, those with WebDDoS (the worst with 0.90), LDAP, Portmap, UD-PLag, NTP, TFTP and NetBIOS attack data. In scenarios with more clients, where these attacks are present in multiple local datasets (although mixed with other attacks), FLAD learns all attacks more accurately, although it consistently produces an F1 Score below 0.92 on the WebDDoS attack.

The convergence time of the FL process is greatly impacted by the global model's ability to perform well on the two TCP-based attacks. By adding new clients as described earlier in this section, we can potentially include more local datasets that contain TCP-based attack data, even if mixed with other types of attacks. This allows the global model to learn more quickly how to correctly classify the TCP-based attacks, which have proven to be critical for the convergence of the FL process (see Section 8.1.1).

9 DISCUSSION

As previously elaborated in Section 4, the FL process can potentially face security threats from malevolent clients and servers. These entities may attempt to exploit their positions within the federation to manipulate the classification performance of the global model or gain insight into the confidential data of other participants.

There are two main differences between FLAD and FEDAVG approaches: (1) FLAD involves clients sharing their accuracy metrics with the server, whereas FEDAVG requires the sharing of local training set sizes, and (ii) with FLAD no test data is required at the server site to assess the quality of the global model. In situations where malicious clients seek to compromise the global model, they can employ conventional techniques such as model poisoning, label flipping, etc. [46] to manipulate the model's performance. This manipulation may result in the model missing certain types of DDoS attacks or other forms of intrusions. This is a general problem of FL that has been tackled in previous studies [13], [34], [35], [36], [37]. To further exacerbate the challenges associated with FL, a malicious client can employ different strategies. One approach involves transmitting fake information on the number of samples to the server (n_k in Equation 1), effectively assigning more weight to its manipulated contributions (FEDAVG). Alternatively, the client can deliberately provide a lower accuracy value, leading to an extended allocation of training rounds and epochs (FLAD). Regarding the latter consideration, it is essential to recognise that a malicious client can independently determine the learning rate or the number of training epochs [47], irrespective of whether the FL process employs FLAD or FEDAVG. However, in the case of FLAD, the client can strategically exploit the accuracy value to secure more training rounds than actually required, thereby influencing the global model's weights in favour of its malicious objectives (e.g., classifying a DDoS type as benign). In contrast, with FEDAVG, this manipulation is not feasible, as client selection is always random.

Clients of FL may face the threat of reconstruction attacks perpetrated by a malicious server, which can exploit information on global model architecture, clients' gradients and other metadata to infer details of the original clients' training data [48], [49], [50]. This inherent vulnerability within FL can become more pronounced in implementations where supplementary information is disclosed to the server. Examples of such information include the count of local training samples (as seen in FEDAVG) or the accuracy of the global model on the local validation sets (as in the case of FLAD). To mitigate this vulnerability, various techniques have been recently proposed in the scientific literature, starting from differential privacy [51], [52], which consists of adding noise to distort the shared parameters, to a novel approach based on obscuring the clients' gradients via fragmentation [49].

In this study, our focus has been on addressing the challenges associated with achieving convergence in the FL process within network intrusion detection scenarios. We recognise that issues of potential malicious clients and servers are crucial factors to consider when implementing an FL framework. However, we acknowledge that these aspects fall outside the scope of our current work. Nevertheless, we consider them as opportunities for further investigation and exploration in the future.

10 CONCLUSIONS

The main challenge in adopting FL techniques in cybersecurity is assessing the performance of the global model on those attacks whose feature distributions are only known by clients. In this paper, we have presented FLAD, an adaptive FL approach for training feed-forward neural networks for DDoS attack detection, that implements a mechanism to monitor the classification accuracy of the global model on the clients' validations sets, without requiring any exchange of data. Thanks to this mechanism, FLAD can estimate the performance of the aggregated model and dynamically tune the FL process by assigning more computation to those clients whose attack profiles are harder to learn. FLAD has been proven to significantly reduce convergence time while also enhancing classification accuracy when compared to current state-of-the-art FL solutions.

We have validated FLAD using an unbalanced dataset of non-i.i.d. DDoS attacks. However, we see the potential of the FLAD's approach in other application domains where clients are expected to contribute with brand new data classes, whose profiles are not available to the server for the assessment of the global model. Although outside the scope of this work, we believe that an interesting research direction could be exploring the adaptability of FLAD to generic Network IDSs in the presence of unknown network attack types, its relevance to host-based IDSs in contexts with zero-day vulnerabilities exploited to compromise computing infrastructure, and its potential portability to other domains such as image classification, where some image classes may be exclusively available in the local datasets of a subset of clients.

101070473 (project FLUIDOS).

REFERENCES

[1] Purplesec. (2021) 2021 cyber security statistics: The ultimate list of stats, data and trends. [Online]. Available: https://purplesec.us/resources/cyber-security-statistics/

[2] D. S. Berman, A. L. Buczak, J. S. Chavis, and C. L. Corbett, "A survey of deep learning methods for cyber security," *Information*, vol. 10, no. 4, p. 122, 2019.

[3] S. A. Salloum, M. Alshurideh, A. Elnagar, and K. Shaalan, "Machine learning and deep learning techniques for cybersecurity: A review." in *AICV*, 2020, pp. 50–57.

[4] 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*, 2017.

[5] D. Lv, X. Cheng, J. Zhang, W. Zhang, W. Zhao, and H. Xu, "Ddos attack detection based on cnn and federated learning," in *2021 Ninth International Conference on Advanced Cloud and Big Data (CBD)*. IEEE, 2022, pp. 236–241.

[6] J. Zhang, P. Yu, L. Qi, S. Liu, H. Zhang, and J. Zhang, "FLDDoS: DDoS Attack Detection Model based on Federated Learning," in *Proc. of IEEE TrustCom*, 2021.

[7] Q. Tian, C. Guang, C. Wenchao, and W. Si, "A lightweight residual networks framework for ddos attack classification based on federated learning," in *Proc. of IEEE INFOCOM Workshops*, 2021.

[8] Roberto Doriguzzi-Corin, "FLAD source code," 2023. [Online]. Available: https://github.com/doriguzzi/flad-federated-learning-ddos

[9] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[10] V. Pourahmadi, H. A. Alameddine, M. A. Salahuddin, and R. Boutaba, "Spotting Anomalies at the Edge: Outlier Exposure-based Cross-silo Federated Learning for DDoS Detection," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[11] J. Li, Z. Zhang, Y. Li, X. Guo, and H. Li, "FIDS: Detecting DDoS Through Federated Learning Based Method," in *Proc. of IEEE TrustCom*, 2021.

[12] M. Dimolianis, D. K. Kalogeras, N. Kostopoulos, and V. Maglaris, "Ddos attack detection via privacy-aware federated learning and collaborative mitigation in multi-domain cyber infrastructures," in *2022 IEEE 11th International Conference on Cloud Networking (CloudNet)*. IEEE, 2022, pp. 118–125.

[13] Z. Yin, K. Li, and H. Bi, "Trusted multi-domain ddos detection based on federated learning," *Sensors*, vol. 22, no. 20, p. 7753, 2022.

[14] S. I. Popoola, R. Ande, B. Adebisi, G. Gui, M. Hammoudeh, and O. Jogunola, "Federated Deep Learning for Zero-Day Botnet Attack Detection in IoT-Edge Devices," *IEEE Internet of Things Journal*, vol. 9, no. 5, pp. 3930–3944, 2022.

[15] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the Mirai Botnet," in *26th USENIX security symposium (USENIX Security 17)*, 2017, pp. 1093–1110.

[16] T. D. Nguyen, S. Marchal, M. Miettinen, H. Fereidooni, N. Asokan, and A.-R. Sadeghi, "Dïot: A federated self-learning anomaly detection system for iot," in *Proc. of ICDCS*, 2019.

[17] Y. Liu, S. Garg, J. Nie, Y. Zhang, Z. Xiong, J. Kang, and M. S. Hossain, "Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach," *IEEE Internet of Things Journal*, 2020.

[18] V. Mothukuri, P. Khare, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and G. Srivastava, "Federated learning-based anomaly detection for iot security attacks," *IEEE Internet of Things Journal*, 2021.

[19] Y. Zhao, J. Chen, D. Wu, J. Teng, and S. Yu, "Multi-task Network Anomaly Detection Using Federated Learning," in *Proc. of SoICT*, 2019.

[20] R. Zhao, Y. Wang, Z. Xue, T. Ohtsuki, B. Adebisi, and G. Gui, "Semi-supervised federated learning based intrusion detection method for internet of things," *IEEE Internet of Things Journal*, 2022.

[21] O. Friha, M. A. Ferrag, M. Benbouzid, T. Berghout, B. Kantarci, and K.-K. R. Choo, "2DF-IDS: Decentralized and Differentially Private Federated Learning-based Intrusion Detection System for Industrial IoT," *Computers & Security*, p. 103097, 2023.

[22] O. Friha, M. A. Ferrag, L. Shu, L. Maglaras, K.-K. R. Choo, and M. Nafaa, "FELIDS: Federated learning-based intrusion detection system for agricultural Internet of Things," *Journal of Parallel and Distributed Computing*, vol. 165, pp. 17–31, 2022.

[23] H. Wang, L. Muñoz-González, D. Eklund, and S. Raza, "Non-iid data re-balancing at iot edge with peer-to-peer federated learning for anomaly detection," in *Proc. of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021.

[24] L. Wang, S. Xu, X. Wang, and Q. Zhu, "Addressing class imbalance in federated learning," in *Proc. of the AAAI Conference on Artificial Intelligence*, 2021.

[25] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[26] M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, "Self-balancing federated learning with global imbalanced data in mobile systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, 2020.

[27] J. Zhang, C. Luo, M. Carpenter, and G. Min, "Federated Learning for Distributed IIoT Intrusion Detection using Transfer Approaches," *IEEE Transactions on Industrial Informatics*, 2022.

[28] C. Briggs, Z. Fan, and P. Andras, "Federated learning with hierarchical clustering of local updates to improve training on non-iid data," in *Proc. of IJCNN*, 2020.

[29] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing Federated Learning on Non-IID Data with Reinforcement Learning," in *Proc. of INFOCOM*, 2020.

[30] S. Ji, W. Jiang, A. Walid, and X. Li, "Dynamic Sampling and Selective Masking for Communication-Efficient Federated Learning," *IEEE Intelligent Systems*, 2021.

[31] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2020.

[32] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "Adaptive federated learning in resource constrained edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205–1221, 2019.

[33] D. Y. Zhang, Z. Kou, and D. Wang, "Fedsens: A federated learning approach for smart health sensing with class imbalance in resource constrained edge computing," in *Proc. of IEEE INFOCOM*, 2021.

[34] Y. Qu, M. P. Uddin, C. Gan, Y. Xiang, L. Gao, and J. Yearwood, "Blockchain-enabled federated learning: A survey," *ACM Computing Surveys*, vol. 55, no. 4, pp. 1–35, 2022.

[35] Z. Zhang, Y. Zhang, D. Guo, L. Yao, and Z. Li, "Secfednids: Robust defense for poisoning attack against federated learning-based network intrusion detection system," *Future Generation Computer Systems*, vol. 134, pp. 154–169, 2022.

[36] Y.-C. Lai, J.-Y. Lin, Y.-D. Lin, R.-H. Hwang, P.-C. Lin, H.-K. Wu, and C.-K. Chen, "Two-phase Defense Against Poisoning Attacks on Federated Learning-based Intrusion Detection," *Computers & Security*, vol. 129, p. 103205, 2023.

[37] H. Lycklama, L. Burkhalter, A. Viand, N. Küchler, and A. Hithnawi, "Rofl: Robustness of secure federated learning," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 453–476.

[38] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Adversarial machine learning attacks and defense methods in the cyber security domain," *ACM Computing Surveys*, vol. 54, pp. 1–36, 2021.

[39] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 2019, pp. 1–8.

[40] University of New Brunswick. (2019) DDoS Evaluation Dataset. [Online]. Available: https://www.unb.ca/cic/datasets/ddos-2019.html

[41] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Software Networking*, vol. 2018, no. 1, pp. 177–200, 2018.

[42] R. Doriguzzi-Corin, S. Millar, S. Scott-Hayward, J. Martinez-del Rincon, and D. Siracusa, "Lucid: A practical, lightweight deep learning solution for ddos attack detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 876–889, 2020.

[43] G. Combs. (2022) Tshark - dump and analyze network traffic. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[44] D. Endres and J. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information Theory*, 2003.

[45] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proc. of the 12th USENIX Conference on Operating Systems Design and Implementation*, 2016.

[46] G. Xia, J. Chen, C. Yu, and J. Ma, "Poisoning Attacks in Federated Learning: A Survey," *IEEE Access*, vol. 11, pp. 10 708–10 722, 2023.

[47] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *International conference on artificial intelligence and statistics*. PMLR, 2020, pp. 2938–2948.

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

[49] S. H. Na, H. G. Hong, J. Kim, and S. Shin, "Closing the Loophole: Rethinking Reconstruction Attacks in Federated Learning from a Privacy Standpoint," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022.

[50] J. Geiping, H. Bauermeister, H. Dröge, and M. Moeller, "Inverting Gradients - How Easy is It to Break Privacy in Federated Learning?" ser. NIPS'20, 2020.

[51] X. Shen, Y. Liu, and Z. Zhang, "Performance-enhanced federated learning with differential privacy for internet of things," *IEEE Internet of Things Journal*, vol. 9, no. 23, pp. 24 079–24 094, 2022.

[52] R. Hu, Y. Gong, and Y. Guo, "Federated learning with sparsified model perturbation: Improving accuracy under client-level differential privacy," *arXiv preprint arXiv:2202.07178*, 2022.