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 the original FL algorithm 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*—Federated Learning, Intrusion Detection, Distributed Denial of Service, Heterogeneous Data**

I. 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 on 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, especially when dealing with zero-day vulnerabilities and attack vectors, due to the complexity and variability of emerging attacks, and the lack of data with relevant and up-to-date attack profiles.

A recent approach to tackle the issues inherent to data and ANN model update is called collaborative learning, which allows multiple independent parties to train/update their Intrusion Detection System (IDS) by sharing information of recent attack profiles. Collaborative learning techniques have started to gain attention in recent years, when McMahan et al. [4] presented the so-called 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.

As analysed by McMahan et al. in their paper, the convergence of the FL process can be challenging for various reasons, spanning from the presence of non-independent and identically distributed (i.i.d.) data across clients, to unbalanced datasets, to unstable communication links between clients and server. Nevertheless, FL has proved to work well in image classification and language modelling tasks, where the aggregated global model can achieve target test-set accuracy in a reasonable number of rounds. However, Federated Averaging (FedAvg), the FL mechanism conceived by McMahan et al., requires the availability of an adequately representative test set at the server side to control the training process. This might limit the applicability of FL to domains where only a subset of the data classes can be tested by the server. For instance, network data of recent cyber incidents against one or more clients might contain sensitive information that cannot be shared with the server for testing. In such a scenario, the server would not be able to verify the performance of the aggregated model with the latest attack traffic.

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) 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 by comparing it against FEDAVG in a worst-case condition, with unbalanced and non-i.i.d. DDoS attack data among the clients. We demonstrate that FLAD improves the FEDAVG algorithm 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, while ensuring 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 original FEDAVG algorithm, demonstrating that FLAD is more efficient and produces aggregated models of higher classification accuracy.
- A prototype implementation of FLAD, publicly available for testing and use [5].

The remainder of this paper is organised as follows. Section II presents the FEDAVG algorithm and highlights its limitations in training models for cybersecurity applications. Section III reviews and discusses the related work. Section IV provides a threat model analysis. Section V presents the FLAD adaptive federated training for DDoS attack detection. Sections VI and VII detail the dataset and the experimental setup. In Section VIII, FLAD is evaluated and compared with FEDAVG. Finally, the conclusions are given in IX.

II. 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 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).

$$w_t \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_t^k \tag{1}$$

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.

There are two main parameters that 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.

A. *Limitations of* FEDAVG

In Section VIII we show that the FEDAVG algorithm, as conceived by McMahan et al., does not satisfy the requirements of DDoS attack detection, such as:

- 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, irrespectively 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.
- Accurate detection of all attack types in realistic conditions, where the detection system must learn from unbalanced and non-i.i.d. training data obtained from heterogeneous DDoS attack types characterised by different traffic rates and feature distributions. The weighted average of

FEDAVG gives more importance to attack profiles with the largest local training sets, to the detriment of the smallest ones. We argue that FEDAVG might fail to learn short attacks with out-of-distribution features.

Finally, we note that FEDAVG assumes that test data is available at 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 learnt the new attack would be using 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 client's sensitive information, with the risk of losing IP, transport and application layer features that could be critical for model validation.

B. 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. As per previous discussion, the solution must satisfy the following constraints:

C1 No training data can be shared among clients or between clients and 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, to 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.

III. RELATED WORK

Implementing a robust and efficient FL system is challenging [6] and often requires application-specific tuning and optimisation. As discussed in Section II, FEDAVG was designed and tested for image classification and language modelling tasks, but it presents a range of drawbacks that may limit its adoption in other contexts, such as network traffic classification for DDoS attack detection. Therefore, in this Section, we review and discuss the current state of the art of FL research, with focus on challenges related to the cybersecurity domain, including efficiency of the FL process, unbalanced and non-i.i.d. data.

A. FL 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, LwResnet [7], FLDDoS [8] and FIDS [9] are three recent solutions for DDoS attack detection evaluated on the CIC-DDoS2019 dataset, the same used to assess the performance of FLAD (cf. Section VI). 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 authors of FLDDoS propose a methodology based on data augmentation to solve the problem of local data imbalance and on a two-stage model aggregation 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 clients' data with the central server. In a recent paper, Popoola et al. [10] 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 [11] 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 [12]–[15] 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 II. Finally, an interesting work by Wang et al. [16] presents a peer-to-peer variation of FL to train a model for anomaly detection in IoT without the need of 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 II-B, 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, some works rely on the vanilla FEDAVG algorithm [9], [10], [12], which aggregates the local models using weighted averaging. We will demonstrate in Section VIII-A 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.

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

The accuracy of ANNs trained with FEDAVG can degrade significantly in scenarios with class imbalance [17] or with non-i.i.d. data [18]. To mitigate the issues of unbalanced data across clients, Duan et al. [19] 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. Briggs et al. [20] 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. [17] 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. [18] 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 [21] improves the performance of FEDAVG on non-i.i.d. data through 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.

C. Efficient FL

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, minimisation of the number of local training rounds assigned to clients.

The approach of Ji et al. [22] 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. [23] 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. [24] and FedSens [25] focus 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 has the flexibility to 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. [24] 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 VIII-A 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, in Section V 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. In Section VIII, we demonstrate that FLAD is robust to model re-training upon 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.

IV. 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. We also assume that neither server nor clients are malicious, thus they do not try to compromise the global model with poisoned data (e.g., weights obtained with mislabelled samples).

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 [26]. 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.

V. METHODOLOGY

FLAD enhances FEDAVG to solve the problem formulated in Section II-B. 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 adjust the amount of computation they perform at each training round. The rationale is that clients with higher scores should train less and vice-versa. In addition, the server can implement a training-stopping strategy that ensures acceptable performance on all the attack types, with no need

TABLE I
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 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 $c \in C$
e_{min}, e_{max}	Minimum and maximum local training epochs
s_{min}, s_{max}	Minimum and maximum local training MBGD steps

for test data of all attacks (not always available at the server side, especially in the case of clients experiencing zero-day attacks). Please note that, in order 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., [7], [8], [23]).

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 I.

The pseudo-code in Algorithm 1 describes the main process executed by the server, which orchestrates the operations of the clients. Algorithm 1 takes as input a global model (w_0) and the set of clients involved into the FL process (C). It runs indefinitely until convergence is reached, as controlled by hyper-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 determine the best global model seen along the process (maximum 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 Mini-Batch Gradient Descent (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 $t = 1$, $C_{t-1} = C_0 = C$, that is, the input set of clients (line 4).

At each round, the server performs the arithmetic mean of the parameters of all clients, including those that have not been involved in the last round of training (line 11). 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 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 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
```

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 is the amount of computation (hence epochs and steps) needed from the client. 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.

The pseudo-code in Algorithm 3 illustrates the local training procedure executed by clients starting from weights and biases of the global model w, 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

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

```
Input: Clients ($C$), accuracy scores ($[a^c]_{c \in C}$), average accuracy score ($a^\mu$)
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:         $\sigma = \frac{\overline{a} - a^c}{\overline{a} - \underline{a}}$                ▷ Scale 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 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
```

descend (lines 7-11) and finally returns the updated model to the server.

VI. The Dataset

FLAD is validated with a recent dataset of DDoS attacks, CIC-DDoS2019 [27], 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 [28]. The benign traffic of the dataset has been generated using the B-profile introduced in [29], 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 II.

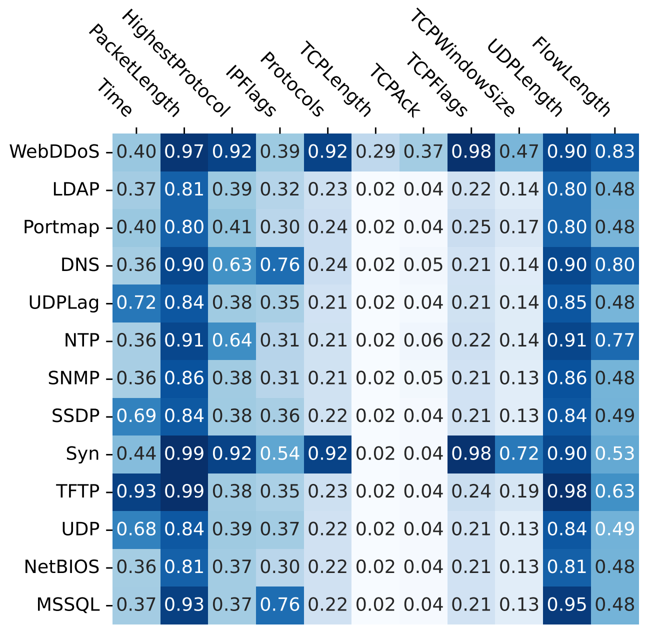
In Table II, the column *#Flows* indicates the amount of bi-directional TCP sessions or UDP streams contained in the traffic traces provided with the dataset, each flow identified by a 5-tuple (source IP address, source TCP/UDP port, destination IP address, destination TCP/UDP port and IP protocol). Before experimenting with our solution, we have pre-processed the traffic traces with the tool developed in our previous work LUCID [30]. 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 [31], 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.

A. 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 Jensen-Shannon Distance (JSD) [32] 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.

TABLE II
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 with 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 of flows each, a few tenths of thousands flows for the LDAP and Portmap reflection attacks, and only 146 flows for the WebDDoS attack.



	Time	PacketLength	HighestProtocol	IPFlags	Protocols	TCPLength	TCPAck	TCPFlags	TCPWindowSize	UDPLength	FlowLength
WebDDoS	0.40	0.97	0.92	0.39	0.92	0.29	0.37	0.98	0.47	0.90	0.83
LDAP	0.37	0.81	0.39	0.32	0.23	0.02	0.04	0.22	0.14	0.80	0.48
Portmap	0.40	0.80	0.41	0.30	0.24	0.02	0.04	0.25	0.17	0.80	0.48
DNS	0.36	0.90	0.63	0.76	0.24	0.02	0.05	0.21	0.14	0.90	0.80
UDPLag	0.72	0.84	0.38	0.35	0.21	0.02	0.04	0.21	0.14	0.85	0.48
NTP	0.36	0.91	0.64	0.31	0.21	0.02	0.06	0.22	0.14	0.91	0.77
SNMP	0.36	0.86	0.38	0.31	0.21	0.02	0.05	0.21	0.13	0.86	0.48
SSDP	0.69	0.84	0.38	0.36	0.22	0.02	0.04	0.21	0.13	0.84	0.49
Syn	0.44	0.99	0.92	0.54	0.92	0.02	0.04	0.98	0.72	0.90	0.53
TFTP	0.93	0.99	0.38	0.35	0.23	0.02	0.04	0.24	0.19	0.98	0.63
UDP	0.68	0.84	0.39	0.37	0.22	0.02	0.04	0.21	0.13	0.84	0.49
NetBIOS	0.36	0.81	0.37	0.30	0.22	0.02	0.04	0.21	0.13	0.81	0.48
MSSQL	0.37	0.93	0.37	0.76	0.22	0.02	0.04	0.21	0.13	0.95	0.48

Fig. 1. JSD map of the probability distributions of the features.

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). With regard to TCP-based attacks, all the *Syn Flood* packets have *Packet Length* equal to 40 bytes, while more than 80% of the *WebDDoS* packets have size of either 66 or 74 bytes.

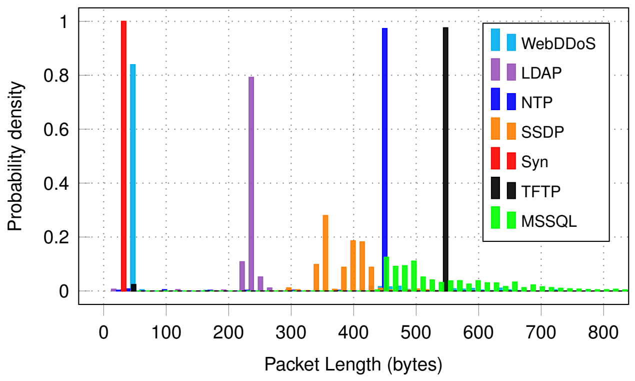


Fig. 2. Probability density functions of the *Packet Length* feature.

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 VIII the negative impact of such out-of-distribution attacks on the convergence of FEDAVG.

VII. Experimental Setup

The FLAD approach is validated using a fully connected neural network model (or Multi-Layer Perceptron (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 V.

As we are interested in measuring the benefits of FLAD over FEDAVG in terms of convergence time and load on the clients (measured as local training time), 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 2.7.0 [33], which means that server and clients are executed on the same machine and communicate through local procedure calls. Please note that this implementation choice has no effect on 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.

A. Dataset preparation

The CIC-DDoS2019 dataset has been distributed to 13 clients, so that each attack is assigned to only one client (pathological non-i.i.d. partition of the data, as defined by McMahan et al. [4]). In addition, we have expressly unbalanced the number of samples across clients, doubling the number of samples from one client to another, starting from the client with the smallest attack (202 WebDDoS samples), to the largest one (MSSQL, reduced to 819204 attack samples). On the other hand, the dataset of each client is balanced (approximately the same number of benign and DDoS samples) and split into training (90%) and test (10%) sets, with 10% of the training set used for validation (Table III).

TABLE III
CIC-DDoS2019 dataset splits.

Attack	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

B. 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. 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 VI). Before processing, each array is re-shaped into a $n \cdot f$-size vector, where packets are lined up one after another in chronological order. The objective of the

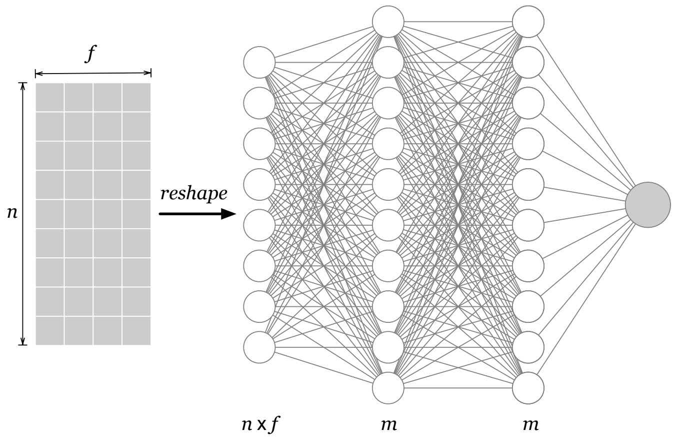


Fig. 3. Architecture of the ANN used to evaluate FLAD.

local training procedure, summarised in Algorithm 3, is to minimise the cross-binary loss function defined in Equation 2. The loss 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.

$$c = -\frac{1}{s}\sum_{j=1}^{s}(y_j \log p_j + (1-y_j)\log(1-p_j)) \qquad (2)$$

In Equation 2, 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 2, tends to 0 when the output probabilities of the flows are close to the respective ground truth labels.

Note that, the loss is computed by each client independently, using only local training data. In the case of non-i.i.d. features across clients (cf. Section VI-A), 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 [18].

C. 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 IV have been used to validate FLAD and to compare it against FEDAVG. The values in the table have been kept constant across all experiments described in Section VIII.

Concerning our implementation of FEDAVG, we use the same MLP architecture used for testing FLAD, while the

TABLE IV
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.

values of epochs and batch size have been chosen based on the performance reported in [4] on both image classification and language modelling tasks. More precisely, we experiment with 1 and 5 epochs/round of local training and with a fixed batch size of 50 samples. Other values could also be used (e.g., 20 epochs/round or batch size of 10 samples), 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.

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 number of hidden layers and activations, we started with larger architectures and then progressively reduced the dimensions until we reached a configuration that allowed a 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 IV. The minimum and maximum values of epochs have been set as the same values used to evaluate FEDAVG. Unlike McMahan et al., 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.

VIII. EXPERIMENTAL EVALUATION

We evaluate various configurations of FLAD on the dataset of DDoS attacks presented in Section VI. As described in Section VII-A, we consider a setup where there is a one-to-one mapping between attack types and clients and where datasets are highly unbalanced across clients. The evaluation focuses on assessing the adaptive approach of FLAD in a realistic cybersecurity scenario, where the federated training must ensure high classification performance of the global model on the attack data of all clients, possibly in a short time. The classification performance of FLAD is measured in terms of *F1 score* (also used as accuracy metric in the implementation of Algorithms 1 and 2 presented in Section V). The *F1 Score* is a widely used metric to evaluate classifier accuracy, computed as the harmonic mean of *Precision* and *Recall*. *Precision* (*Pr*) represents the ratio between the correctly detected DDoS samples and all the detected DDoS samples. *Recall* (*Re*) represents the ratio between the correctly detected DDoS samples and all the DDoS samples in the dataset. The *F1 Score* is formally defined as $F1 = 2 \cdot \frac{Pr \cdot Re}{Pr + Re}$.

A. Convergence analysis

In the first experiment, FLAD is compared against FEDAVG, i.e., the original procedure for federated learning proposed by McMahan et al. [4], where the set of clients that perform local training is randomly chosen at each round and both training epochs and batch size are fixed. Our implementation of FEDAVG is used as a performance baseline. The goal is to expose the limitations of FEDAVG 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.

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 13 clients. Then, we train FEDAVG until the global model reaches approximately the same F1 score as FLAD (we monitor the performance of FEDAVG by gathering the F1 Score of the global model from the clients using the FLAD's mechanism detailed in Algorithm 1). In the case of FEDAVG, we repeat the experiment twice, once with one local epoch/round and once with 5 epochs/round of local training. We remind that with FEDAVG the local training is configured with a static batch size of 50 samples.

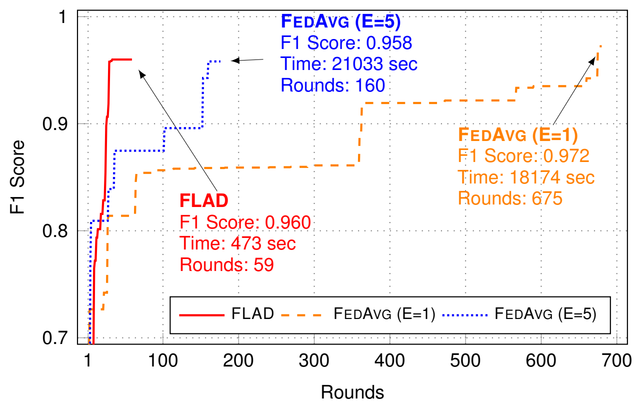


Fig. 4. Comparison between FLAD and two configurations of FEDAVG.

The results of the experiment are plotted in Figure 4, which shows the performance trend of the best aggregated model as obtained up to a given round. Performance is expressed in terms of average F1 score across clients' validation sets.

From the figure, we can clearly see the main benefits of FLAD over FEDAVG: faster convergence in a lower amount of communication rounds. Indeed, FLAD implements a strategy to dynamically select clients at each round based on the performance of the current aggregated model on their local datasets. Thus, the federated training process focuses more on the attacks which are harder to learn, namely the two out-of-distribution attacks WebDDoS and Syn Flood, which are selected more often for local training (40 and 41 rounds out of 59 respectively, compared to an average of around 17 rounds for the other attacks). On the contrary, with FEDAVG, clients are chosen in a random fashion, leading to a higher number of communication rounds necessary to achieve similar accuracy as FLAD. This is clearly shown in Figure 5, which compares FLAD and FEDAVG in terms of performance trend of the global model on out-of-distribution data (namely, the WebDDoS and Syn Flood attack traffic).

Furthermore, FLAD dynamically tunes the amount of computation assigned to each of the selected clients for one round of training, which greatly reduces the average local training time per round (from around 8 sec/round with FLAD, to 27 and 131 sec/round with the two configurations of FEDAVG) and, consequently, the total federated training time. We remind that FEDAVG assigns a fixed number of epochs and MBGD steps to the selected clients.

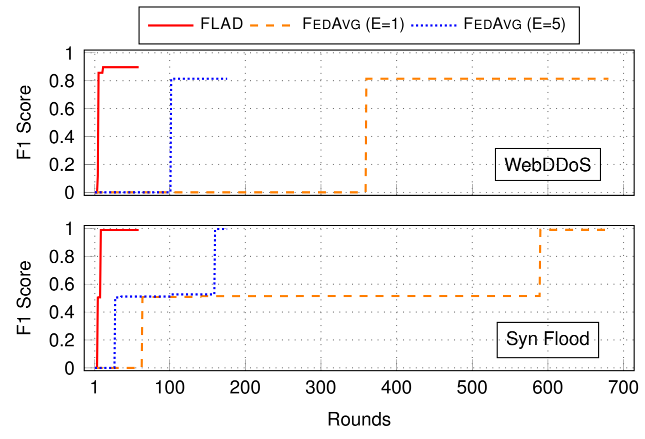


Fig. 5. Performance of FLAD and FEDAVG on out-of-distribution data.

The key improvement of FLAD over to FEDAVG 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. With regard to 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.

B. Federated re-training with new attack data

We now compare FLAD and FEDAVG in a realistic scenario, where the global model needs frequent retraining to learn new attack types.

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. Of course, such a stopping procedure is needed for both FLAD and FEDAVG to advance from step to step. To this aim, like for the previous experiment, we have integrated into FEDAVG the mechanism that monitors the performance of the global model on the local clients' validation sets.

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.

TABLE V
COMPARISON BETWEEN FLAD AND TWO CONFIGURATIONS OF FEDAVG. TIME EXPRESSED IN SECONDS.

Metric	**FLAD**	**FEDAVG**	
		E=1	**E=5**
FL Rounds	605	523	508
Time/Round	1.86	6.00	27.75
Total Time	1125	3089	13840
F1 Score	0.9853	0.8315	0.8657
F1 StdDev	0.0118	0.2564	0.1995

Table V reports average metrics across 10 experiments. *FL Rounds* is the average total number of rounds necessary to reach convergence with all 13 DDoS attack types (starting from 2, as described in the intro of this Section). *Time/Rounds* is the average time taken to complete a single round with clients training in parallel. *Total time* is the average time taken to complete an entire experiment through all its steps (from 2 to 13 attacks). *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; as before, this is 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.

We can see in Table V that, despite FLAD needs more rounds to achieve convergence, the time/round is much lower thanks to the adaptive selection of clients and computation assigned to each client. In this regard, the values in Table V confirm the analysis presented Section VIII-A. The F1 Score and F1 StdDev values reported in Table V demonstrate that FLAD achieves higher classification accuracy during all the steps of the experiment (from 2 to 13 attacks) and across all the attack types. On the contrary, the accuracy of FEDAVG is penalised by the poor performance on small and out-of-distribution attack profiles. This aspect is discussed in the next

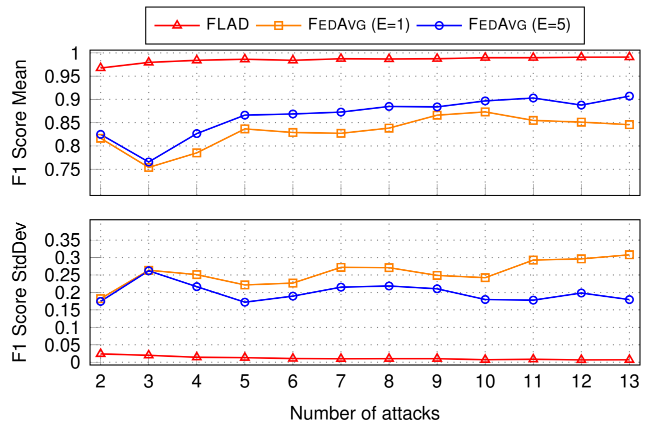


Fig. 6. Mean and standard deviation of the F1 score at increasing number of DDoS attacks.

Section, where we evaluate the resulting models on the local test sets (unseen data).

A performance comparison is also shown in Figure 6, which plots the mean and the standard deviation of the F1 score measured at each step of the experiment. As we can observe in Figure, FedAvg produces poor performance with any combinations of attacks, namely low average F1 score and high standard deviation over the attacks. As also discussed above, the main cause of the degradation is the low classification score of the global model on attacks with out-of-distribution features. On the other hand, FLAD maintains a stable F1 score on all the attack types, as demonstrated by the negligible standard deviation measured at every step of the test.

C. Evaluation on unseen data

We evaluate, on unseen data, the aggregated models obtained with the federated training described in Section VIII-B. Thus, we test the capability of the aggregated models to correctly recognise the 13 attacks. To this aim, Table VI reports the average True Positive Rate (TPR) measured on the test sets of the clients using the final models obtained at the end of the 10 experiments. While FLAD produces very high TPR values with most attacks, this experiment confirms that FedAvg does not perform well on out-of-distribution (o.o.d.) attack traffic (the TCP-based attacks WebDDoS and Syn Flood), and with small training sets, such as those of the WebDDoS, LDAP, Portmap and UDPLag. This confirms the analysis of the aggregated metrics reported in the previous section.

D. Ablation analysis

FLAD implements methods for adaptive selection and configuration of clients, including methods for the dynamic tuning of local training epochs and MBGD steps. In this experiment, we analyse the impact of these methods on the performance of FLAD by replacing them with static parameters. In particular, we repeat the experiment described in Section VIII-B with static epochs E=1,5 and a static batch size of B=50 samples, i.e. static MBGD steps (indicated as S=∞ in this Section, following the notation used by McMahan et al. in their paper).

TABLE VI
Comparison on clients' test sets (TPR).

Attack	FLAD	FedAvg E=1	FedAvg E=5
WebDDoS	0.9136	**0.0727**	**0.6364**
LDAP	0.9889	0.9222	0.9306
Portmap	1.0000	0.9058	0.9183
DNS	0.9872	0.7799	0.7852
UDPLag	0.9978	0.9969	0.9978
NTP	0.9770	0.9567	0.9468
SNMP	0.9992	0.9477	0.9578
SSDP	0.9982	0.9973	0.9958
Syn	0.9865	**0.2762**	**0.3243**
TFTP	0.9949	0.9513	0.9242
UDP	0.9987	0.9986	0.9968
NetBIOS	0.9983	0.9085	0.9282
MSSQL	0.9983	0.9863	1.0000
Global	0.9977	0.9443	0.9557

TABLE VII
Comparison between FLAD configurations using combinations of Adaptive Epochs (E=A), Adaptive Steps (S=A), Static Epochs (E=1,E=5), Static Batch Size (S=∞). Time expressed in seconds.

Metric	E=A S=A	E=1 S=A	E=1 S=∞	E=5 S=A	E=5 S=∞
FL Rounds	605	671	599	624	654
Time/Round	1.86	1.28	2.98	2.46	11.32
Total Time	1125	853	1781	1528	7396
F1 Score	0.9853	0.9819	0.9813	0.9862	0.9899
F1 StdDev	0.0118	0.0155	0.0167	0.0104	0.0097

Table VII reports the results obtained with FLAD as described in Section V (column E=A,S=A), including adaptive selection of clients, adaptive local training epochs and MBGD steps. In the other tests, we keep the adaptive selection of clients, but we use static epochs combined with static or adaptive MBGD steps. In the Table, the metrics are the same as in Table V.

From the results reported in the Table, we can observe a lower convergence time with the adaptive configuration of the number of MBGD steps (columns S=A). Indeed, adaptive MBGD steps lead to much shorter rounds, hence to a reduction of the total training time of 52% and 79% on average with E=1 and E=5 respectively, compared to settings with static batch size (S=∞).

Among the configurations with adaptive MBGD steps, that with E=1 reaches the best trade-off between accuracy and convergence time. In comparison, fully-adaptive FLAD (E=A,S=A) adds more computation to the clients (we recall that E=A means that the server assigns to the clients a number of training epochs ranging between 1 and 5) with the expected drawback of increasing the total training time. However, we can see that, by enabling adaptive epochs, FLAD produces a better classification accuracy and stability across all the attack types in a lower number of rounds.

IX. 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 introduced 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. We have demonstrated that the proposed approach greatly reduces convergence time and improves classification accuracy.

Our analysis of FLAD's performance has shown that the dynamic tuning of the local MBGD steps is very effective in reducing the convergence time up to almost 80%, compared to configurations with static batch sizes. On the contrary, we have observed a negative impact on the duration of the single training rounds when FLAD is configured with dynamic local training epochs. Therefore, in application scenarios where the computing resources devoted to model training are limited, one epoch of local training per round is probably the best solution, even considering the slight detriment of the classification accuracy caused by this choice.

We have validated FLAD using an unbalanced dataset of non-i.i.d. DDoS attacks. However, we are confident that the proposed approach can be effectively adopted in other cybersecurity applications where clients are expected to contribute with zero-day attack samples, whose profiles are not available to the server for the assessment of the global model.

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, pp. 1273–1282.

[5] Roberto Doriguzzi-Corin, "FLAD source code," 2022. [Online]. Available: https://github.com/doriguzzi/flad-federated-learning-ddos

[6] 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.

[7] Q. Tian, C. Guang, C. Wenchao, and W. Si, "A lightweight residual networks framework for ddos attack classification based on federated learning," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021, pp. 1–6.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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*, vol. 8, no. 8, pp. 6348–6358, 2020.

[14] 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.

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

[16] 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 *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2021, pp. 153–163.

[17] 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.

[18] 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.

[19] 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.

[20] 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 International Joint Conference on Neural Networks (IJCNN)*, 2020.

[21] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing Federated Learning on Non-IID Data with Reinforcement Learning," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020.

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

[23] 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.

[24] 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.

[25] 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 *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, 2021, pp. 1–10.

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

[27] 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.

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

[29] 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.

[30] 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.

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

[32] D. Endres and J. Schindelin, "A new metric for probability distributions," *IEEE Transactions on Information Theory*, vol. 49, no. 7, 2003.

[33] 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.