Multi-Model-based Federated Learning to Overcome Local Class Imbalance Issues

Jiefei Liu*, Huiping Cao*, Abu Saleh Md Tayeen*, Satyajayant Misra*, Pratyay Kumar* and Jayashree Harikumar†
*New Mexico State University, Las Cruces, New Mexico, USA
Email: {jiefei, hcao, tayeen, misra, pratyay}@nmsu.edu
†DEVCOM Analysis Center, WSMR, New Mexico, USA
Email:jayashree.harikumar.civ@army.mil

2023 International Conference on Machine Learning and Applications (ICMLA) | 979-8-3503-4534-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICMLA58977.2023.00044

***Abstract*—Federated learning (FL) is gaining much popularity in designing Intrusion Detection Systems (IDS) due to its ability to maintain data privacy and reduce communication costs. Existing FL-based IDS are generally tested with balanced class distribution for all clients where each client has data with all attack traffic categories. This is a very strong assumption. In reality, we often encounter a local class imbalance issue, which means that each client only has traffic with a few number of attack types. This issue creates a critical challenge in FL by leading to poor model performance and convergence. Several studies have made efforts to solve this issue through the clustering of local model parameters. However, their methods are costly and require either prior knowledge or training to select the number of clusters. In this work, we propose a Multi-Model-based Federated Learning (*MMFL*) framework, which automatically groups the local models of clients having similar class distribution, and a novel data augmentation method to add instances with unknown attack types to the datasets of local devices. Our extensive experiments with two large latest intrusion detection datasets show that *MMFL* outperforms the five baselines on the intrusion detection task.**

***Index Terms*—Multi-model federated learning, Intrusion detection, Local class imbalance, Clustering.**

I. Introduction

Federated Learning (FL) [1] has gained significant attention as a promising alternative to the centralized Machine Learning (ML) architecture to address communication overhead and data privacy concerns. In the network Intrusion Detection Systems (IDS), many researchers [2]–[5] have utilized FL to develop IDS. In a typical FL-based IDS, each network device uses its traffic dataset to train a local intrusion detection model and sends only the local model parameters to the central server for aggregation and subsequent dissemination back to clients. This process is generally iterated to produce an improved global model that can detect malicious network traffic.

Although many proposed FL-based IDSes have shown improved performance over preliminary solutions, they rely on a strong assumption that client devices have uniform class distribution in their dataset. That means, every client has data for all attack traffic categories or types. This assumption is not realistic. In the real world, due to the distributed geographic locations of clients, the class distribution is often imbalanced. In extreme cases, some clients may not have samples for certain attack categories. This issue termed as *local class imbalance* poses a significant challenge in FL by hindering the global model convergence and degrading its performance.

Few researchers modified the aggregation strategy [6] or proposed a new loss function [7] to mitigate the impact of the local class imbalance issue. However, these methods try to implicitly deal with the problem rather than eliminating it. Some works [8]–[11] attempted to group the clients with the same data distribution utilizing the L2-based distance of local model parameters and generate multiple global models. However, their methods either need pre-training iterations or pre-defined optimal value as a parameter to select the number of clusters, which is an impractical FL scenario. Moreover, these methods utilize all the model parameters to perform the clustering, which may not generate effective client groups and can be computationally expensive. Some works [10], [11] employ multiple models for multi-task learning instead of solving the class imbalance issue of single-task learning.

To address the above-mentioned issues, in this paper, we propose a Multi-Model-based FL framework, *MMFL*, for a single-task FL-enabled system such as network intrusion detection. In contrast to the existing methods, *MMFL* trains multiple global models by aggregating local models of clients belonging to each cluster. The final prediction of an instance utilizes the votes of global models, each of which represents a similar class distribution. For each client, we augment the instances with a special type of instance to accommodate the attack types that do not occur in that specific client.

The contributions of our paper are as follows:

1) Our proposed framework, *MMFL*, automatically groups clients and train multiple global models. The grouping algorithm utilizes only partial local model parameters of clients. This framework also incorporates an online option to group clients dynamically in every round that allows clients to drop and join.
2) *MMFL* offers a novel data augmentation strategy which introduces *unknown-attack* traffic to each client to account for class imbalance issue in different local devices.
3) We evaluated *MMFL* on two common and most recent network intrusion detection datasets: CICDDoS2019 [12] and CICIDS2017 [13] by creating class imbalanced partitions and compared its performance with five baselines.

1946-0759/23/$31.00 ©2023 IEEE
DOI 10.1109/ICMLA58977.2023.00044

The remainder of this paper is organized as follows: We provide an overview of related works in Section II. Section III presents the details of our framework, *MMFL*. In Section IV, we describe the data, our experimental setup, and the results. Finally, in Section V, we conclude with future works.

II. Related Work

Several researchers [2]–[5] proposed FL-integrated IDS to strengthen IoT network security. These works assumed that the datasets of all clients are balanced in terms of classes and quantity. However, in a real-world scenario, the network devices that act as FL clients may not have the same class distribution, which is denoted as the *local class imbalance*. We note that some works (*e.g.,* [14], [15]) build FL systems to protect against malicious clients. Such studies showed that the global model might fail to converge or produce incorrect predictions when the data of malicious clients is heterogeneous (non-iid). Our work focuses on addressing the *local class imbalance* issue, instead of considering the existence of malicious clients.

Several researchers attempted to address this issue by proposing methods that only try to reduce the impact of the local class/data imbalance on the performance of FL. For instance, Hsu *et al.* [6], modified the FedAvg [1] by adding momentum to the server and proposed FedAvgM. The authors in [7] designed an online FL monitoring method to track the imbalance in FL systems globally and provide a new loss function to mitigate the impact of the global imbalance.

Other research works [8]–[11], [16] focused on clustering clients or using multiple models to deal with class imbalance problems in FL. Duan *et al.* [8] proposed FedGroup which groups the clients using a cosine similarity-based metric and federally trains each group model. Bhuyan *et al.* [11] extended the FedAvg by proposing three algorithms for selecting clients and assigning multiple models to learn multiple uncorrelated tasks simultaneously. Ghosh *et al.* [9] proposed IFCA framework to identify the cluster membership of each client and optimize the cluster models in a distributed setting. In the FeSEM [10], the authors used K-means clustering with L2 distance to group the clients' models and performed multi-center aggregation. Cao *et al.* [16] created several global models through a random selection of clients to mitigate the effects of malicious clients. The final predictions are obtained through the use of majority voting from multiple global models.

In all the cluster-based approaches [8]–[10], the number of clusters needs to be known at the beginning of the FL process. Finding the optimal value of clusters requires completing a full FL training which is costly and impractical. Other studies [10], [11] resort to multi-task learning, which either trains multiple uncorrelated models or treats the model personalization for every node as a different task. Unlike the mentioned literature that uses all the weights in the client models, our framework designs a simple yet effective clustering method to group the clients based on partial information of the client models.

III. *MMFL* Framework

We first describe our major technical contributions in *MMFL* design in Section III-A. Then we provide an overview of *MMFL* and the details of its training and testing procedures.

A. Multi-model Design and Novel Data Augmentation Method

Our designed framework particularly wants to overcome the *local class imbalance* issue. We design two novel strategies.

The first strategy is that our framework generates multiple global ML models. This is one important characteristic of *MMFL*. This characteristic makes it different from other traditional FL frameworks (*e.g.,* vanilla FL framework [1]), which generates only one global model. Since datasets across clients in FL are imbalanced (*i.e.,* there is a variation in the number of samples for different classes/labels), training only one global model worsens its convergence and performance. Our intuition behind the design of *MMFL* is that clients in FL can be partitioned into different clusters or groups. Clients with similar class distribution belong to the same group. Learning a global ML model for each group can alleviate the divergence problem and improve the classification performance. Furthermore, our MMFL designs a new clustering strategy to group clients using partial information in the clients' local models (Section III-C), which is different from other multi-model-based FL methods (Section II).

The second strategy is a novel data augmentation method. When the distribution of the attack traffic in the clients is different, a model built on a specific client caters towards the data distribution in that client. A model built for a dataset without an attack type (*e.g.,* $A1$) performs very poorly in the prediction stage when the testing instance is of this attack type (*e.g.,* $A1$). To make these models to accommodate predicting instances with all possible attack types, we introduce a new type of fake attack traffic, called *unknown attack*. For client k, we add instances with this *unknown attack* traffic and let U_k represent the set of such instances. When the client id is unknown, we use U to represent this instance set for a general client. Instances in U_k are of attack types that do not exist in the original local dataset of client k. For example, given that the whole system can process five different attack types ($A1 - A5$), a local device has instances for normal traffic and traffic of attack types $A1$, $A2$, and $A3$. Using this strategy, the updated dataset should include instances from attack types $A4$ and $A5$ and those instances are put in U and labeled as *unknown attack* instances.

B. Overview of MMFL

MMFL is devised to operate in networks with devices such as routers and security gateways that are considered as clients. The clients are mainly responsible for collecting relevant data and training local ML models. We assume (as most FL frameworks do) that there exists one central server which coordinates the training process by clustering clients, distributing the global ML models to the corresponding clients, receiving model updates from the clients, and performing

per cluster aggregation of the received models. After the training is completed, the produced global models are used during deployment to detect anomalous network traffic. Our *MMFL* framework is illustrated in Figure 1.

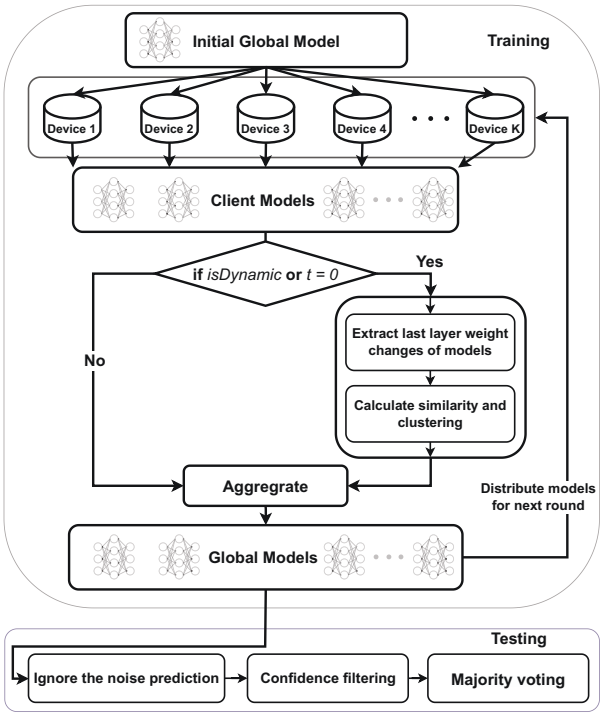


Fig. 1: Overview of *MMFL* framework (static and dynamic).

C. *Training of* MMFL

Algorithm 1 presents the training procedure of our framework, *MMFL*. The algorithm takes two parameters as input: i) the total number of training rounds, T; and ii) *isDynamic:* a boolean value to decide whether the clustering of clients is **static** (*i.e.,* perform clustering only in the first round) or **dynamic** (*i.e.,* perform clustering in every round). **Dynamic** clustering allows handling of clients that may drop due to connection issues during the FL training. The training in Algorithm 1 produces a set $\mathcal{G}$ of global models as output.

The training of *MMFL* starts on the server side in Step 1. In Step 2, the server initializes a single global model, W_0. In Steps 3 to 6, the server assigns each client a copy of this global model, W_0. Here, W_t^k denotes the global model assigned to the k-th client at round t. In Step 4, the server shares with all the clients a public dataset that contains data for all class labels. This dataset is utilized to create U instances (*i.e.,unknown attack*) for each client.

Steps 7 to 23 represent the training process of *MMFL* for T rounds. In each training round, t, the server first allots two sets, $\mathcal{W}$ and $\mathcal{G}$ to store the local models of the clients and the global models respectively (Step 8). The server then distributes the allocated global models to their respective clients and collects the locally trained models in Steps 9 to 12 by invoking the *LocalTrain* function. When *LocalTrain* function is invoked for the k-th client at round t, the client trains its assigned global model, W_t^k with its local dataset. Here, w_t^k (Line 10) denotes the locally trained model for the k-th client at round t. If the clustering strategy is dynamic (*i.e., isDynamic* is true) or this is the first round of training (*i.e.,* $t = 0$), the server clusters the local models of the clients into R groups (Function *HeuristicCluster*) in Step 14.

Algorithm 1: *MMFL* Training Procedure

Input: i) $T(int)$: total number of training rounds;
ii) *isDynamic* $(bool)$: whether the clustering is static or dynamic
Output: A set, $\mathcal{G}$ of global models

1 **Function** `Server()`**:**
2 Initialize a global model, W_0
3 **for** $k \leftarrow 1$ **to** K *clients* **do**
4 Send a dataset used to generate *unknown-attack* instances to client k.
5 $W_0^k \leftarrow W_0$
6 **end for**
7 **for** $t \leftarrow 0$ **to** T *rounds* **do**
8 $\mathcal{W} \leftarrow \emptyset$, $\mathcal{G} \leftarrow \emptyset$
9 **for** $k \leftarrow 1$ **to** K **do**
10 $w_t^k \leftarrow LocalTrain(k, W_t^k, e)$
11 $\mathcal{W} \leftarrow \mathcal{W} \cup w_t^k$
12 **end for**
13 **if** *isDynamic* ***or*** $t = 0$ **then**
14 $\{C_1, C_2 \cdots C_R\}$ = *HeuristicCluster* ($\mathcal{W}$, δ)
15 **end if**
16 **for** $j \leftarrow 1$ **to** R **do**
17 $W_j^g \leftarrow FedAvg(C_j)$
18 $\mathcal{G} \leftarrow \mathcal{G} \cup W_j^g$
19 **end for**
20 **for** $j \leftarrow 1$ **to** R **do**
21 $W_{t+1}^k \leftarrow W_j^g$, $\forall k$ if $w_t^k \in C_j$
22 **end for**
23 **end for**
24 **return** $\mathcal{G}$

One major difference of our method compared with other existing multi-model approaches is the strategy (*Heuristic-Cluster*) we designed to cluster (or group) the clients. First, to compute the two models' similarity in the grouping process, we did not utilize all the values of the hidden features. Instead, for each model, we extract the weight changes of its last layer as its features. We used such features based on a theorem in [7], which states that if two models' last layers are linear, the gradients of weights of the last layer are the same when the class labels of the input data are identical. Second, we did not use existing clustering algorithms such as K-means to group the clients. Instead, we designed a simple but effective heuristic strategy. We randomly choose a client C as a seed for one group and grow this group by adding other clients who do not belong to any existing group and whose model similarity with C is above a threshold δ until no client can be added to this group. Then, we randomly choose another client who does not belong to any group yet and grow it using the same procedure for the next group. This heuristic process terminates when all the clients belong to some group. In reality, the similarity threshold, $\delta = 0.8$ can help generate good grouping which is consistent with our

experimental setting. Our experiments show that setting the similarity threshold is much easier when we use the weight changes of the last layer compared with using all the weights.

In Steps 16 to 19, the server applies the FedAvg [1] aggregation operation on the models in each cluster and generates R global models. Here, W_j^g denotes the global model produced from the j-th cluster. The server then assigns the new global models to the corresponding clients for round $t+1$ training in Steps 20 to 22.

D. Making Predictions using MMFL

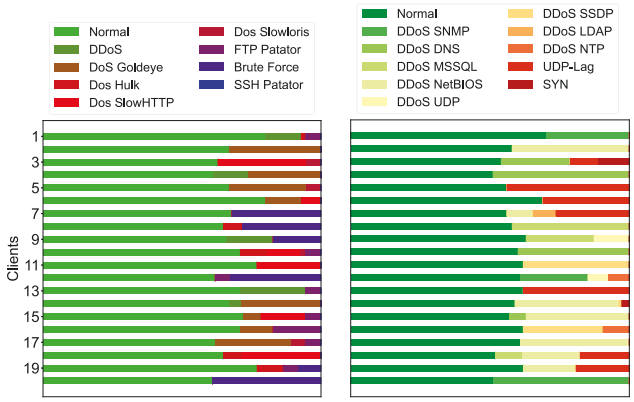
The set of multiple global models, $\mathcal{G}$, is used to make predictions for forthcoming traffic. Each global model can predict an instance to be normal, unknown, or a specific type of attack traffic. For example, given an instance, a global model g_1 can predict it to be normal, an unknown attack, A_1, or attack A_2, while another global model g_2 can predict it to be normal, an unknown attack, A_2, or attack A_3. If a model predicts an instance to be unknown attack or the prediction (to normal or a specific attack) has low confidence, this model is ignored when aggregating the predictions. The final prediction decision of an instance comes from the majority voting of the remaining global models. If all the global models predict an instance to be unknown traffic, the final prediction is an unknown attack (which is a wrong prediction).

IV. Experiments

A. Data

1) Datasets: We selected the commonly used intrusion detection datasets CICDDoS2019 [12] and CICIDS2017 [13] for our experiments. Considering the extensive scale (millions of records) of the original CICDDoS2019 dataset, we followed an approach similar to other studies [17] that selected 41 useful features and drew 10,000,000 samples. The samples include 5,000,000 instances of normal traffic flows and 5,000,000 instances of ten different DDoS attack traffic flows with each class having an equal size. We split this dataset into three subsets: i) 50% of the dataset as training, ii) 40% of the data for generating *unknown attack* instances, and iii) the remaining 10% of the data for testing.

Similar to the other studies [18], we preprocessed the CICIDS2017 dataset and selected 40 useful features. We removed instances of three classes such as 'Heartbleed', 'Infiltration', and 'Bot' from the dataset because they had an overly low number of instances. The centralized model was not able to learn with too few samples and after the partition the clients did not get sufficient instances to train their local models for those classes. Instances of classes like 'FTP Patator' and 'SSH Patator' were generated using the same brute force function. Therefore, we combined instances of those classes and labeled them as a 'brute-force' class. Finally, for the CICIDS2017 dataset, we got 2,203,723 instances of normal traffic flows and 467,261 instances of eight different attack traffic flows. Since this dataset is smaller in size compared to the CICDDoS2019 dataset, to avoid under-fitting, we split 60% of it for training, 30% of it to generate *unknown-attack* instances for all the clients, and 10% of it for testing.



(a) CICIDS2017 (b) CICDDoS2019

Fig. 2: Data and class distribution for 20 partitions

2) Extreme Class Imbalance Partition: In order to create a realistic IDS simulation, the client should expect to see only a few types of attacks with the majority of traffic being normal. Thus our partition contains less than three types of attacks, and the normal traffic outweighs the attack traffic. As a result, there may be issues with local data and class imbalance. Figure 2 demonstrates the class distribution across 20 partitions for both CICIDS2017 and CICDDoS2019 datasets. The green label represents normal traffic, while other colors indicate different types of attack traffic. Due to the limited number of attack instances for CICIDS2017, each partition has more normal traffic.

B. Experimental Settings

We performed all the experiments on a server with Intel Xeon 5220R 2.20 G CPUs, 512 GB RAM, and 4 Nvidia RTX A4000 GPUs with 16 GB memory per GPU. The Flower framework [19] was utilized for running FedAvg and FedAvgM, and we implemented the models using PyTorch. We made our code publicly available[1].

1) Models: To evaluate the performance of the proposed methods, we chose the multi-class classification task, which categorizes the normal traffic and different kinds of attack traffic flows. We selected the Multi-Layer Perceptron (MLP) as the clients' model that had one input layer, two fully connected layers with hidden unit sizes of 64 and 128, respectively, and a softmax output layer.

2) Parameters: In all the experiments, we set the default number of clients to 20. We used Adam optimizer with a learning rate, $\eta = 0.01$ to train the models. We set the default number of epochs e for local model training in each client to be 10, We limited the maximum training round, T, to 20 for all FL methods. To enhance clustering quality, in the first round of Algorithm 1 (*i.e.,* $t = 0$), we set the value of e larger (five times the default e value) than the other rounds.

[1]https://github.com/JiefeiLiu/FL_Multi_model

Methods	CICDDoS2019		CICIDS2017	
	Accuracy	***F1***	***Accuracy***	***F1***
Centralized	0.837 ± 0.002	0.656 ± 0.014	0.963 ± 0.002	0.867 ± 0.003
NonFed	0.990 ± 0.024	0.960 ± 0.098	0.927 ± 0.039	0.707 ± 0.105
FedAvg	0.590 ± 0.022	0.204 ± 0.032	0.845 ± 0.014	0.317 ± 0.042
FedAvgM	0.630 ± 0.050	0.249 ± 0.079	0.872 ± 0.010	0.419 ± 0.044
FeSEM	0.579 ± 0.009	0.155 ± 0.011	0.846 ± 0.004	0.329 ± 0.018
Random (Static)	0.595 ± 0.006	0.229 ± 0.015	0.838 ± 0.006	0.259 ± 0.043
Random (Dynamic)	0.608 ± 0.011	0.249 ± 0.020	0.853 ± 0.001	0.313 ± 0.037
MMFL (Static)	$\mathbf{0.813 \pm 0.003}$	$\mathbf{0.586 \pm 0.007}$	$\mathbf{0.944 \pm 0.009}$	$\mathbf{0.766 \pm 0.024}$
MMFL (Dynamic)	$\mathbf{0.793 \pm 0.009}$	$\mathbf{0.559 \pm 0.012}$	$\mathbf{0.943 \pm 0.005}$	$\mathbf{0.792 \pm 0.010}$

TABLE I: Performance comparison with CICDDoS2019 and CICIDS2017 datasets

After extensive experiments, we set the threshold, $\delta = 0.8$ for the similarity threshold.

3) Metrics: To evaluate our method, we utilized the performance metrics defined as follows,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, F1 = \frac{2 \times Pre \times Rec}{Pre + Rec}.$$

Here, Pre = $\frac{TP}{TP+FP}$, Rec = $\frac{TP}{TP+FN}$, and TP/FP (True/False Positive) represents the number of instances correctly/wrongly classified as an attack; TN/FN (True/False Negative) represents the number of instances correctly/wrongly classified as normal. The reported $F1$ value is the macro average of the $F1$ for all the class labels.

C. Baselines

As references, we implemented two models.

i) *Centralized*. This method trains one model centrally using the data from all the clients. Note that this model is not for FL. We use this method as a reference which provides the upper bound of performance for all FL-based algorithms.

ii) *NonFed*. This method does not use any FL framework. It trains an MLP model for each client with their local data and the model is tested using instances with attack types existing in their local data.

We also compared our framework, *MMFL*, with the following baseline models.

- *FedAvg* [1]. This method uses the conventional FL framework to produce a single global model by aggregating a large fraction of the client models. The experiments set the fraction to be 90%.
- *FedAvgM* [6]. This method modifies the FedAvg by adding momentum during aggregation at the server and aggregates one global model. In our experiments, the fraction is set to 90% and the momentum is set to 0.7.
- *FeSEM* [10]. This method utilizes K-means clustering to group local models and create global models. This model computes the L2 distance between weights from all layers of the local models to determine similarity. In our experiments, the number of clusters is set to 4.
- *Random (Static)*. This FL method is similar to our method. The difference is that instead of using clustering, the server randomly selects the clients to assign different global models in the first round. Then, it uses the same clients' assignment for the rest of the training rounds. More concretely, when the FL training starts, the server will initialize $|\mathcal{G}|$ global models, where $|\mathcal{G}|$ is a parameter. In our experiments, we set $|\mathcal{G}| = 5$ for this method.
- *Random (Dynamic)*. This FL method is exactly the same as the *Random (Static)* method except that it randomly selects the clients for global models in every training round. We set $|\mathcal{G}| = 5$ for this method.

D. Performance Comparison

We ran each experiment five times on both datasets and present the accuracy and F1-score metric values using $M \pm S$ form in Table I where M and S represent the mean and the standard deviation of those values respectively.

Table I shows the results. Our framework *MMFL* (both **static** and **dynamic** versions) achieves the highest accuracy and $F1$ score for both the CICDDoS2019 and CICIDS2017 datasets. For example, when tested on CICDDoS2019 dataset, the accuracy of *MMFL* (static) is generally more than 30% higher than the other baselines. The improvement on the $F1$ metric is even more significant. *MMFL* (static) achieves a dramatic improvement with an $F1$ score that almost doubles (CICIDS2017) and triples (CICDDoS2019) the $F1$ scores of all the other methods.

Note that the FedAvgM is designed to reduce the impact of data/class imbalance. Thus, it can improve the FedAvg, as shown in Table I. However, our *MMFL* method still dramatically outperforms the FedAvgM method.

The *NonFed* baseline shows the overall best performance which denotes that the MLP model fits with local training data reasonably well. Note that this is not a FL method. Each client's model is not tested against any instance of attack types not in that client's training data. This is an ideal but not realistic setting, thus the good performance cannot be matched in a real FL setting. The *Centralized* baseline provides an upper bound for all the FL methods.

Among all the FL methods, *MMFL* is the only method that achieves comparable performance to the *Centralized* method.

E. Impact of Parameters

We conducted experiments to assess the effect of the number of clients and the size of unknown attack traffic on the performance of our framework.

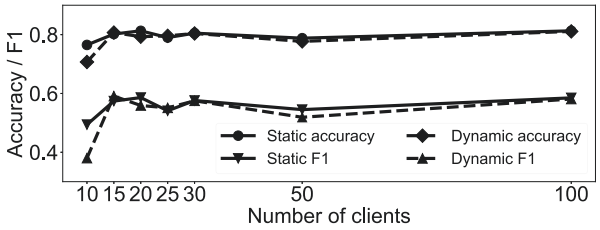
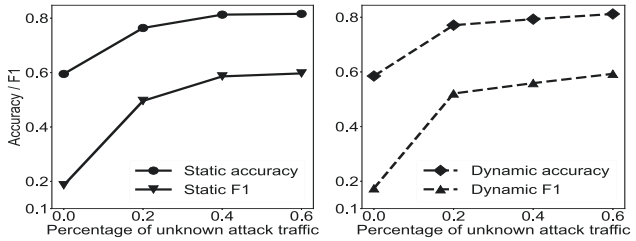


Fig. 3: Performance comparison with different numbers of clients on CICDDoS2019 dataset

1) Impact of varying the number of clients: In this experiment, we used the CICDDoS2019 dataset and varied the number of clients from 10 to 100 because existing works [2]–[5] generally used 10-20 clients in FL for IDS setting. In Figure 3, we depict the performance (*i.e.,* accuracy and $F1$) of *MMFL* (static) and *MMFL* (dynamic) with an increasing number of clients. From Figure 3, we found that with ten clients, the *MMFL* (static) and the *MMFL* (dynamic) obtained 5% and 12% lower accuracy compared to the scenario where there are 20 clients. This is because with the ten-client scenario each cluster may only have one model to aggregate and this will make it difficult for the global model to converge. As the number of clients increases, the performance of *MMFL* becomes stable.



(a) Static framework (b) Dynamic framework

Fig. 4: Performance comparison with different unknown-attack sample size on CICDDoS2019 dataset

2) Impact of different unknown attack sample size: The size of the *unknown attack* traffic U for specific clients affects the model performance. We vary the percentage of *unknown attack* traffic instances over the total number of attack instances in each client k. The results in Figure 4 show that the performance improves with the increase in the size of U. However, we cannot keep increasing the size of U just to improve the prediction accuracy because the dataset that is sent to each client to generate *unknown attack* instances (Line 3 in Algorithm 1) may not be sufficient to sample a large U. Therefore, selecting an appropriate *unknown attack* sample size for clients is crucial. After conducting extensive experiments, we have determined that an *unknown attack* sample size of 0.4 balances the two factors well.

V. Conclusions

This paper introduced a multi-model-based FL framework, *MMFL* and a new data augmentation strategy to facilitate the federated learning process in IDS for scenarios where clients do not have instances of all attack types. Our framework creates multiple global models for different groups of clients determined by a simple yet effective clustering algorithm. It utilizes the data augmentation method to address the local class imbalance issue in clients. Our extensive experiments on FL-based IDS tasks have demonstrated that *MMFL* outperforms the baselines. In the future, we aim to enhance the robustness of MMFL by exploring alternative data sharing methods from server to client for data augmentation and considering the existence of malicious clients.

Acknowledgment

This work has been sponsored by the DEVCOM Analysis Center and accomplished under Cooperative Agreement Number W911NF-22-2-0001. This work has also been partially funded by the Department of Energy, DE-SC0023392 and NSF #1914635.

References

[1] McMahan Brendan, Moore Eider, and et al. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.
[2] Noor Ali Al-Athba Al-Marri, Bekir S Ciftler, and Mohamed M Abdallah. Federated mimic learning for privacy preserving intrusion detection. In *IEEE (BlackSeaCom)*, pages 1–6, 2020.
[3] Li Jianhua, Lyu Lingjuan, and et al. FLEAM: A federated learning empowered architecture to mitigate DDoS in industrial IoT. *IEEE Transactions on Industrial Informatics*, 18(6):4059–4068, 2021.
[4] Qiaofeng Qin, Konstantinos Poularakis, Kin K Leung, and et al. Line-speed and scalable intrusion detection at the network edge via federated learning. In *IFIP Networking Conference*, pages 352–360. IEEE, 2020.
[5] Viraaji Mothukuri, Prachi Khare, Parizi Reza M, and et al. Federated-learning-based anomaly detection for IoT security attacks. *IEEE Internet of Things Journal*, 9(4):2545–2554, 2021.
[6] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.
[7] Wang Lixu, Xu Shichao, and et al. Addressing class imbalance in federated learning. In *AAAI*, volume 35, pages 10165–10173, 2021.
[8] Moming Duan and et al. Fedgroup: Efficient federated learning via decomposed similarity-based clustering. In *IEEE ISPA/BDCloud/SocialCom/SustainCom*, 2021.
[9] Avishek Ghosh, Jichan Chung, and et al. An efficient framework for clustered federated learning. *NeurIPS*, 33:19586–19597, 2020.
[10] Long Guodong and et al. Multi-center federated learning: clients clustering for better personalization. *WWW*, 26(1):481–500, 2023.
[11] Neelkamal Bhuyan and Sharayu Moharir. Multi-model federated learning. In *Int'l Conf. on COMSNETS*, pages 779–783. IEEE, 2022.
[12] Iman Sharafaldin, Arash Habibi Lashkari, Saqib Hakak, and Ali A Ghorbani. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In *ICCST*, pages 1–8. IEEE, 2019.
[13] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.
[14] Cao Xiaoyu and et al. Provably secure federated learning against malicious clients. In *AAAI*, volume 35, pages 6885–6893, 2021.
[15] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to Byzantine-Robust federated learning. In *29th USENIX Conf. on Security Symposium*, pages 1605–1622, 2020.
[16] Sai Praneeth Karimireddy and et al. Byzantine-robust learning on heterogeneous datasets via bucketing. *arXiv:2006.09365*, 2020.
[17] Hussain Faisal, Abbas Syed Ghazanfar, and et al. IoT DoS and DDoS attack detection using ResNet. In *INMIC*, pages 1–6. IEEE, 2020.
[18] Osama Faker and Erdogan Dogdu. Intrusion detection using big data and deep learning techniques. In *ACM SE Conf.*, pages 86–93, 2019.
[19] Daniel J Beutel, Taner Topal, and et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.