MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM) | 979-8-3503-2181-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/MILCOM58377.2023.10356272

FLNET2023: Realistic Network Intrusion Detection Dataset for Federated Learning

Pratyay Kumar*, Jiefei Liu*, Abu Saleh Md Tayeen*, Satyajayant Misra*, Huiping Cao*,
Jayashree Harikumar†, and Oscar Perez†
*New Mexico State University, Las Cruces, New Mexico, USA
Email: {pratyay, jiefei, tayeen, misra, hcao}@nmsu.edu
†DEVCOM Analysis Center, WSMR, New Mexico, USA
Email: jayashree.harikumar.civ@army.mil, oscar.a.perez46.civ@army.mil

***Abstract*—Developing new defensive mechanisms to respond to evolving security threats is crucial for enforcing network security. Recently, defenses leveraging Federated Learning (FL) have become prominent in Intrusion Detection Systems (IDS) to incorporate the surging growth and distributed nature of the network infrastructure. To evaluate these FL-based IDSes and to achieve better detection performance, researchers commonly perform equal and balanced partitions of the existing popular datasets, such as NSL-KDD, UNSW-NB15, and CICDDoS2019, among clients. However, partitions of these datasets are not representative of the class-imbalanced scenarios found in real-world networks where each client may possess different categories of attack traffic with an uneven number of instances in their dataset. Moreover, they overlook the fundamental data distribution property in a network environment where data must be associated with individual IP addresses, particularly the destination IPs that are identifiable as FL clients. To fill these gaps, we introduce a novel dataset, Federated Learning for Networks (*FLNET2023*), which is strategically generated by gathering data from network traffic across ten unique routers within a real-world network topology emulated using the CORE tool. We also evaluate the *FLNET2023* using two FL aggregation algorithms and compare its performance against the latest intrusion detection dataset, CICDDoS2019, with traditional partitions to demonstrate the challenges of FL-based IDSes on realistic datasets.**

***Index Terms*—Network intrusion detection system, federated learning, DoS, DDoS, Web Attacks, Infiltration Attacks.**

I. Introduction

As the Internet expands, the increasing variety of cyber-attacks necessitates the use of an Intrusion Detection System (IDS) that monitors network traffic and detects signs of such attacks. Recently, combining Federated Learning (FL) [1] with IDS has gained much popularity because of their ability to train machine learning (ML) models across multiple decentralized nodes/devices while preserving device privacy and reducing communication costs [2]–[4]. Since there are no designated datasets for evaluating FL-enabled IDSes, the existing prominent network intrusion detection datasets have been extensively used for FL experiments. For example, the NSL-KDD [5], the UNSW-NB15 [6], the CICIDS2017 [7], and the CICDDoS2019 [8] datasets, have been widely adopted due to their contemporary collection of network traffic scenarios, encompassing a more comprehensive range of attack types.

However, most of these existing datasets are becoming dated. Further, while evaluating FL-enabled IDSes [2]–[4], these datasets have been equally partitioned (*i.e.,* data are divided across multiple clients such that each client has an equal number of data instances) and balanced (*i.e.,* the number of data instances belonging to different classes are the same) for the convenience of better performance. Nonetheless, these partitions do not accurately represent the real-world networks' class-imbalanced nature, where each device, depending on its location in the network, may gather different types of attack traffic with an unequal number of instances in their dataset. This may pose significant challenges, including performance degradation and convergence problem for the global model in FL-enabled IDSes. Additionally, in contemporary centralized dataset partitions, traffic data are not mapped to the IP addresses of the devices that are considered clients for the FL-based IDS environment. As a result, the partitioning process leads to data being improperly and incorrectly allocated to different devices/routers, which distorts the genuine distribution of network traffic. While intentional unequal and imbalanced partitions from existing datasets may mimic the class imbalance property, they lack realistic mapping and authentic distribution of real-world networks.

To address these gaps in FL-enabled IDSes, in this paper, we introduce a strategy of creating datasets by presenting *FLNET2023* that can better capture real-world network traffic scenarios in a distributed environment and thus is more suitable for FL experiments. The contributions of our paper are as follows:

1) We utilized the Common Open Research Emulator (CORE) [9], a tool developed by the U.S. Naval Research Laboratory, to design the testbed configuration with real-world network topology, simulate diverse normal traffic, and implement various attack types. We chose the CORE to take advantage of its ability to replicate real-world network behaviors accurately.
2) We collected the data in a distributed manner over ten routers to reflect the realistic FL scenario for network environments. We labeled our dataset and made it publicly available.
3) We evaluated our dataset with two FL aggregation

algorithms and compared our results with those of the traditional partitions of the CICDDoS2019 dataset in an FL setting. We also discussed the challenges of using our dataset in FL-based IDS through experimental results.

The remainder of this paper is organized as follows: We present an overview of related work in Section II. Section III elaborates on our process of benchmark dataset generation using the CORE with details of its characteristics. In Section IV, we evaluate our dataset in FL scenario and discuss the implications of our results. Finally, in Section V, we conclude by summarizing our findings and suggesting future work.

II. Related Work

FL has attracted significant interest in recent years due to its distributed learning nature and the ability to secure the privacy of user data. Several researchers utilized FL for intrusion detection in networks. Al-Marri *et al.* [2] proposed an FL-based IDS by integrating mimic learning with FL to prevent reverse engineering attacks on clients' models. They equally distributed the NSL-KDD [5] dataset among ten clients to run the FL experiments. Rahman *et al.* [3] proposed an FL-based IoT intrusion detection scheme and compared it to the centralized and self-learning approaches. They experimented with three different data distribution scenarios: i) data distribution per attack type, ii) equal data distribution of all attack types, and iii) random data distribution of all attack types of the NSL-KDD [5] dataset among four clients. Li *et al.* [10] utilized FL-empowered architecture to mitigate DDoS attacks in industrial IoT devices. They ran simulations by randomly distributing the UNSW-NB15 [6] dataset among four devices to test their method. Qin *et al.* [11] combined Binarized Neural Network and FL to design a framework to detect attacks from incoming network packets. They split the CICIDS2017 [12] among several gateway nodes of the network. Zhang *et al.* [4] proposed FLDDoS, an FL-based DDoS attack detection model. They shuffled and unevenly divided the CICDDoS2019 [8] dataset among 21 clients to show the detection performance of FLDDoS.

The existing intrusion detection datasets used by the works mentioned above were collected centrally and were not intrinsically designed for the FL. Moreover, in all of the FL experiments the datasets were partitioned in a way that ignores the class imbalance that occurs in the real-world networks owing to each node having insight obtained based only on its data and the proper allocation of data to its owners. Therefore, to overcome these shortcomings, in this paper, we introduce a novel intrusion detection dataset that is collected using a system to emulate distributed environment and is more suitable for testing FL-based IDSes.

III. *FLNET2023* Dataset Generation

In this section, we outline the methodology involved in generating our benchmark dataset, *FLNET2023*, which is publicly available with accompanying necessary documentation[1]. We first describe the real-world network topology used in our data-gathering procedure. Second, we delve into the generation of both normal and attack traffic using various tools. Third, we discuss the feature extraction and labeling process of our dataset. Finally, we describe the characteristics of our dataset.

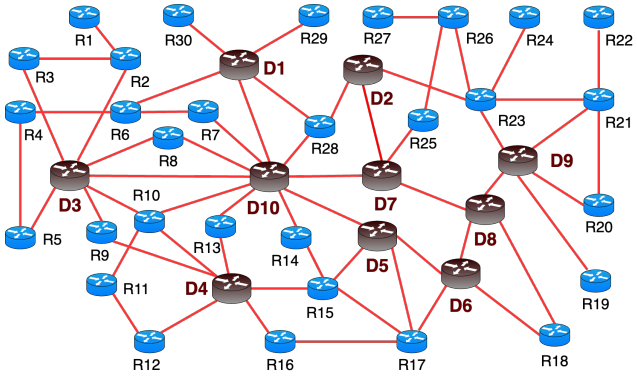

Fig. 1: Network Topology

A. Network Topology

We chose Europe's GEANT-2012 network topology, a renowned research and education network obtained from the Internet Topology Zoo [13] to mimic a realistic and complex environment, allowing the collection of our dataset. This real-world topology, depicted in Figure 1, symbolizes a comprehensive large-scale network infrastructure similar to those in Internet Service Providers (ISPs) with 40 nodes, each representing primary routers serving specific geographical locations or functions.

We replicated the GEANT-2012 topology within the emulator, CORE [9] as it provides the essential functionality of accurately emulating real-world network dynamics, making it an indispensable tool for our data collection procedure. In this emulated environment, we collected network traffic data from a subset of ten strategically chosen router nodes identified by $\{D1, D2, D3, ..., D10\}$ as illustrated in Figure 1. These nodes effectively cover different network regions maintaining multiple connections. By choosing such nodes, we ensured a broader, more representative dataset that captured various network behaviors and conditions.

B. Traffic and Attack Orchestration

Using CORE and the topology in Figure 1, we generated normal traffic as well as attack traffic from various categories, including Denial of Service (DoS), Distributed Denial of Service (DDoS), Web attacks, and Infiltration attacks. The choice of these attack types was motivated by their common occurrence in real-world scenarios, their potential for significant harm, and the diversity in their attack patterns, which present unique challenges in detection. To generate the

[1]Dataset-URL: https://github.com/nsol-nmsu/FML-Network

normal traffic and to orchestrate each attack traffic category, we employed the most commonly used and up-to-date open-source tools that allow different traffic configurations. We detail the traffic generation process for each type below.

1) Normal traffic: We generated normal traffic using the hping3[2] tool. We utilized several representative types of network protocols, such as TCP, UDP, and ICMP, and considered a variety of devices and traffic patterns.

2) DoS attack: We generated attack traffic from two types of DoS attacks: *DoS Hulk* and *DoS SlowHTTP*, each with a unique attack strategy. To orchestrate these attacks, we employed the Hulk[3] and the SlowHTTPTest[4] tools. The primary objective of these attacks is to deplete server resources, thereby disrupting service access for legitimate users. In our emulation, nodes such as $R2, R4, R21$, and $R28$ in Figure 1 were the servers.

3) DDoS attack: To orchestrate DDoS attacks at different layers of the OSI (Open Systems Interconnection) model, we utilized the MHDDoS[5] tool that facilitated the emulation of Layer-4 TCP and Layer-7 attacks, including BOT, STOMP, and DYN. The primary objective of these attacks is to inundate the network services with a flood of traffic, causing service disruptions for legitimate users. In our emulation, we set up victims' systems using Python's built-in HTTP server on routers $R6, D9, D5$, and $D10$ as shown in Figure 1.

4) Web attack: To emulate web attacks, we targeted common threats, including SQL Injection (SQLi), Cross-site scripting (XSS), and Command Injection. These threats were orchestrated using specific tools: sqlmap[6] for SQLi, xsstrike[7] for XSS attacks, and commix[8] for Command Injection. To emulate a realistic attack scenario, we deployed Flask-based web applications on routers $R13, R14$, and $R28$ and set up PostgreSQL databases on these routers with specific vulnerabilities that are exploited by the orchestrated web attacks which can lead to severe consequences, such as data leakage.

5) Infiltration attack: We chose to orchestrate a man-in-the-middle (MITM) attack for this category as it allows one to stealthily gain sensitive information, making it a particularly dangerous network threat. The orchestration of this attack was performed using the bettercap[9] tool, which facilitates ARP spoofing. The essence of this attack lies in intercepting and potentially altering or monitoring communications between two entities without detection. In our emulation, the attackers were workstations connected to different routers.

C. Features and Labels

To construct our dataset, we utilized the command-line interface of the Wireshark tool[10] to capture network traffic in the form of Packet Capture (PCAP) files. Most of these PCAP files serve as a comprehensive record of raw traffic data. For example, 1 Gb PCAP file generated by *DDoS TCP* attack houses an exact copy of every byte from over 3,675,374 packets on the network. To ensure precise labeling of data for different types of attacks, packets were captured separately for each attack scenario.

After successfully capturing the raw network traffic, we next extracted flow-based features from the PCAP files. For this task, we employed the CICFlowMeter[11], a robust flow analyzer tool. It facilitated the extraction of 82 distinct features. The set of features for each flow extracted from every PCAP file was stored in separate CSV files for further analysis. The last phase of our process involved labeling the flow-based data in the CSV files. We carefully executed this task with custom scripts, ensuring each attack traffic flow data point was accurately associated with its corresponding attack category. We generated separate CSV files for each attack category, which removed the chances of errors and ensured proper labeling.

D. Dataset Characteristics

Our dataset, *FLNET2023*, presents the distributed nature of networks and maintains the correct allocation of flows to each node because we collected the data from ten different routers positioned at distinct locations. Moreover, our dataset illustrates the class-imbalanced property of real-world scenarios where each device may encounter different categories of attacks and can have an unequal number of data instances. For instance, in our dataset, router $D1$ contains 205,215 data instances representing four types of attack traffic: *DDoS Bot* (87,808), *DoS SlowHTTP* (10,590), *Infiltration-MITM* (2341), *Web-Command Injection* (330), and Normal traffic (104,146). On the other hand, router $D6$ captures mostly *DoS Hulk* attacks with 404,462 instances, intermixed with 207,480 instances of Normal traffic. We show the combined statistics of each traffic category across all nodes for our training and testing sets in Table I.

Label	Train	Test
Normal	2,368,998	189,836
DDoS Bot	87,808	22,667
DDoS Dyn	322,920	15,341
DDoS Stomp	389,560	92,286
DDoS TCP	1,861,672	302,081
DoS Hulk	1,644,381	108,345
DoS SlowHTTP	97,706	12,092
Infiltration MITM	30,222	1,870
Web Command Injection	330	345
Web SQL Injection	441	221
Web XSS	3,069	1,533

TABLE I: Overall class distribution of *FLNET2023* dataset

[2]Hping3, https://www.kali.org/tools/hping3/
[3]Hulk, https://github.com/grafov/hulk
[4]SlowHTTPTest, https://github.com/shekyan/slowhttptest
[5]MHDDoS, https://github.com/MatrixTM/MHDDoS
[6]Sqlmap, https://github.com/sqlmapproject/sqlmap
[7]XSStrike, https://github.com/s0md3v/XSStrike
[8]Commix, https://github.com/commixproject/commix
[9]Bettercap, https://www.bettercap.org/
[10]Tshark, 2019. https://tshark.dev/
[11]CICFlowMeter, 2017. https://github.com/ISCX/CICFlowMeter

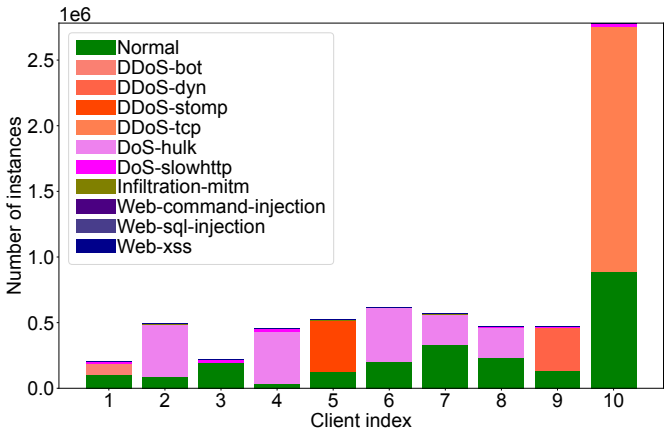

Fig. 2: Class distribution of the *FLNET2023* per client

Figure 2 illustrates an important characteristic of our dataset. It highlights the significant discrepancy in the distribution of instances for various attack classes across all nodes/clients. The disparity in the number of instances is particularly noticeable when comparing classes such as the Web-based SQL injection and command injection to classes like *DDoS Stomp*, *DDoS TCP*, and *DDoS Hulk*. The former classes contain substantially fewer instances, rendering them less visible in the overall bar graph. For instance, the total number of instances in SQL injection, and command injection classes across all clients are 441 and 330, respectively, as compared to the *DDoS Stomp* and *DDoS TCP* classes, which contain 389,540 and 1,861,596 instances respectively.

Unlike the CICDDoS2019 dataset, we narrowed our focus on DDoS attacks by choosing a few subcategories of this attack category. This decision was made for convenience, driven by the primary aim of our research: to generate a realistic and class-imbalanced dataset. In reality, not all types of DDoS attacks will occur with equal frequency at the same time. Therefore, we chose to orchestrate only the *DDoS Dyn*, *DDoS Stomp*, *DDoS Bot*, and *DDoS TCP* attack categories, which are some of the most prevalent and impactful in real-world scenarios. Simultaneously, recognizing the diverse threats in real-world networks, our dataset extends beyond DDoS attacks including a broader range of attack types, such as web-command-injection, web-sql-injection, and an infiltration attack in the form of a man-in-the-middle attack.

IV. Evaluation

A. Experiment Setup

To evaluate the dataset, we opted for a multi-class classification task in the FL settings aiming to detect either normal traffic or different categories of attack traffic. We chose a Multi-Layer Perceptron (MLP) model to do the classification in our experiments. Our MLP model comprises four layers (including three hidden layers) with hidden layer sizes 64, 128, and 64, respectively. We used a Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.01 for training. To construct the FL setup for our emulated data, we considered our data-collecting routers as clients and assumed the presence of a central router or node to function as the server. We compared the performance of the classification task on our dataset and the commonly used CICDDoS2019 [8] dataset. Considering the vast scale (millions of records) of the original CICDDoS2019 dataset, we followed an approach similar to other studies, drawing a sample of 10,000,000 instances from this dataset. This included 5,000,000 instances representing normal traffic flow and 5,000,000 instances embodying ten distinct DDoS attack flows. We allocated 80% of this data for training and the remaining 20% for testing. To create the FL scenario for the CICDDoS2019 dataset, we divided its training data using different partition methods to distribute it among ten network devices/clients. We elaborate on these data partition methods below.

1) Equal and Balanced (EB): This partitioning method is widely used in [2], [3]. In this method, **Equal (E)** means all the clients get the same number of instances and **Balanced (B)** let each partition keep the same attack data distribution as the original dataset. For example, suppose a centralized dataset consists of 100 normal traffic and 100 attack traffic instances (representing three attack types) with attack data distribution ratios of 2:2:1. If the dataset is distributed to two clients, the total number of instances in each client is 100, where 50 instances are of normal traffic and 20, 20, and 10 instances are of the three different attack traffic, respectively.

2) Equal and Imbalanced (EI): In this case, the training data size for each client is the same. However, the **Imbalanced (I)** property randomly produces the clients' attack data distribution. Thus, the data distribution ratios will not remain the same as the original dataset [3], [11].

3) Unequal and Imbalanced (UI): Here, **Unequal (U)** means the training dataset size of each client is not the same. This data partition method divides the dataset into random-sized subsets and generates different data distribution from the original dataset [3], [4].

We performed our FL experiments using the Flower framework [14] on a server machine with Intel Xeon 5220R 2.20G CPUs, 512GB RAM, and 4 Nvidia RTX A4000 GPUs. We implemented the Multi-Layer Perceptron (MLP) model using the PyTorch library[12].

B. Aggregation Algorithms

In this paper, we utilized two FL aggregation algorithms, namely Federated Averaging (FedAvg) [1] and Federated Averaging with server Momentum (FedAvgM) [15] to evaluate our dataset. In the FedAvg, the server updates the weights using $w_{t+1} \leftarrow w_t - \Delta w$. Here, w_t represents the weight of the global model at communication round t, w_{t+1} represents the weight updated by the FedAvg at round $t+1$, and Δw represents an aggregate of the weights from the clients. However, due to the presence of class imbalanced data at

[12]PyTorch, 2019. https://pytorch.org/

Partition	Performance Metrics					
	Aggregation	***Accuracy***	***Precision***	***Recall***	***F1***	***Training time (min)***
Centralized	-	0.837 ± 0.002	0.733 ± 0.004	0.704 ± 0.003	0.665 ± 0.025	485.17 ± 7.671
CICDDoS2019 (EB)	FedAvg	0.836 ± 0.000	0.730 ± 0.001	0.701 ± 0.001	0.664 ± 0.006	202.998 ± 11.349
	FedAvgM	0.835 ± 0.001	0.732 ± 0.005	0.700 ± 0.003	0.656 ± 0.020	210.076 ± 9.031
CICDDoS2019 (EI)	FedAvg	0.835 ± 0.000	0.729 ± 0.000	0.700 ± 0.001	0.661 ± 0.003	202.657 ± 7.081
	FedAvgM	0.834 ± 0.002	0.725 ± 0.007	0.698 ± 0.005	0.665 ± 0.008	200.901 ± 5.799
CICDDoS2019 (UI)	FedAvg	0.835 ± 0.001	0.730 ± 0.002	0.700 ± 0.003	0.667 ± 0.015	219.499 ± 6.180
	FedAvgM	0.835 ± 0.001	0.729 ± 0.001	0.701 ± 0.002	0.662 ± 0.007	231.949 ± 17.414
FLNET2023 dataset	FedAvg	0.682 ± 0.007	0.285 ± 0.049	0.366 ± 0.004	0.277 ± 0.008	239.410 ± 3.104
	FedAvgM	0.812 ± 0.009	0.420 ± 0.075	0.458 ± 0.042	0.428 ± 0.061	243.303 ± 6.028

TABLE II: Performance comparison between CICDDoS2019 and *FLNET2023* dataset

the clients, the FedAvg algorithm suffers from performance degradation as well as convergence issues.

To address these issues, Hsu *et al.* introduced the FedAvgM algorithm. Unlike the FedAvg, the FedAvgM incorporates momentum during the accumulation of model updates by the server, thereby accelerating network training and reducing the impact of data/class imbalanced issues from the clients. To add momentum at the server, the FedAvgM [15] computes v using $v \leftarrow \beta v + \Delta w$ and update the global model with $w_{t+1} \leftarrow w_t - v$, where the momentum $\beta \in [0, 1]$, and v represents an all-ones matrix with the same dimension as w. If we set β as zero, it will correspond to exactly the FedAvg. After tuning with several values, we set $\beta = 0.7$ for FedAvgM in our experiments.

C. Metrics

To evaluate our dataset on the multi-class classification task, we utilized the performance metrics defined as follows, $Accuracy = (TP + TN)/(TP + TN + FP + FN)$, $Precision(Pre) = TP/(TP + FP)$, $Recall(Rec) = TP/(TP+FN)$, $F1 = (2 \times Pre \times Rec)/(Pre+Rec)$, where TP (True Positive) represents the number of instances correctly classified as an attack; TN (True Negative) represents the number of instances correctly classified as normal; FP (False Positive) represents the number of instances incorrectly classified as an attack; FN (False Negative) represents the number of instances incorrectly classified as normal.

D. Results and Discussion

We performed experiments with all three data partitions (defined in Section IV-A) of the CICDDoS2019 dataset and our dataset, *FLNET2023*, using both FedAvg and FedAvgM aggregation algorithms. We ran each experiment three times and showed the performance metric values in $M \pm S$ form in Table II where M represents the average and S represents the standard deviation of the recorded metrics. From Table II, we observe that the performance of both aggregation algorithms on EB, EI, and UI partitions is very similar. All of them can reach around 83% accuracy and 66% F1-score.

Table II includes the results of the centralized scenario (as a reference, Row 1 in the table) where 80% of the unpartitioned CICDDoS2019 dataset was used for training, and 20% of the dataset was used for testing the global model in the central server node. The FedAvg algorithm in the centralized scenario obtained 83.7% accuracy and 66.5% F1-score. The performance of the FL strategies on different types of partitions (EB, EI, UI) on the CICDDoS2019 dataset is very close to the centralized scenario. These results show that the EB, EI, and UI do not affect the performance of the global ML model trained with the FL aggregation algorithms. However, all these data partition methods are not realistic as they assume that all clients have all attack categories. In the real-world scenario, as exhibited by *FLNET2023*, each client may have different classes of attacks with an unequal number of instances. This property may cause performance and model convergence issues.

The last two rows of Table II show the results of both FL aggregation algorithms on our *FLNET2023* dataset. We observe that the FedAvg and the FedAvgM achieved 57% less F1-score and 34% less F1-score on *FLNET2023* respectively compared to those of the EB partition of the CICDDoS2019 dataset. This performance degradation is caused by the fact

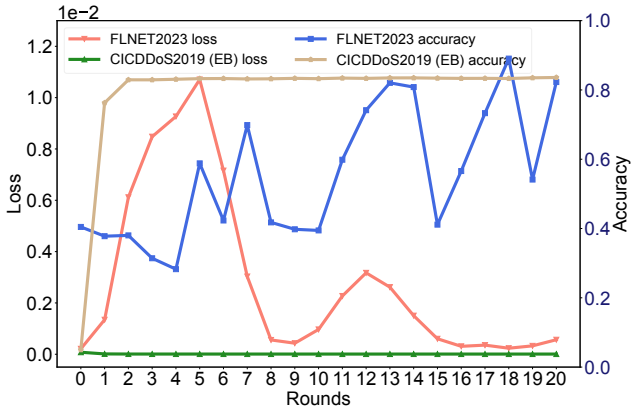

Fig. 3: Loss/Accuracy of FedAvgM for the global model on CICDDoS2019 (EB) and *FLNET2023* dataset

that the global model on the *FLNET2023* does not converge. To further demonstrate this issue, we plot Figure 3 to show the loss/accuracy changes of the global model. Here, the X-

axis represents the rounds, the left Y-axis represents the loss value, and the right Y-axis represents the accuracy. The figure shows that for the EB data partition of the CICDDoS2019 dataset, the global model converged and learned fast (the loss value has been stable since round 3). However, the global model did not converge when we used *FLNET2023* in FL settings. Both the loss value and the prediction accuracy values fluctuate with the communication rounds. This indicates that the local models do not guide the global model in the correct convergence direction.

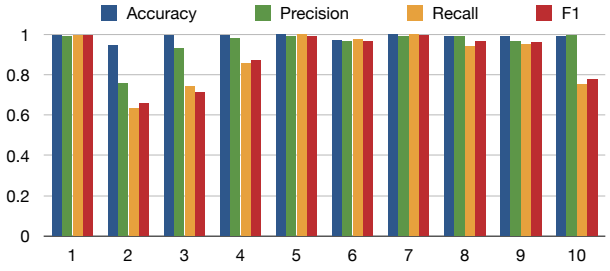

Fig. 4: Local model experiments for *FLNET2023* dataset

To find the rationale for the underwhelming performance of the FL aggregation algorithms over the *FLNET2023*, we performed experiments with each client in a centralized scenario. That means we allowed each client to train their models individually using their respective class-imbalanced local dataset. We then utilized the corresponding class data extracted from the testing set to evaluate their trained models. As illustrated in Figure 4, the accuracy and F1-score for all clients' models exceeded 90% and hovered around 80%, respectively. However, the *FLNET2023* dataset, when tested in an FL-based scenario, exhibited an obvious contrast. As per Table II, the FedAvg achieved merely around 68.2% accuracy and 27.7% F1-score. Although the FedAvgM did better, achieving 81.2% accuracy and 42.8% F1-score, it still fell short compared to the performance of the single client models. That means the FedAvgM does, to some extent, mitigate the impact of data/class imbalanced issues, yet the overall performance remains significantly lower. By analyzing the results from Figure 3 and 4, it's clear that the primary contributor to *FLNET2023*'s underperformance is the class imbalance problem.

Thus, our experiments highlight the challenges faced by FL-enabled IDS, which include performance degradation and convergence issues due to the local class imbalance nature of clients. In a real-world FL-based network environment, it's quite possible for each device to experience no intrusion at all or only observe a limited variety of attack classes. This situation can result in class imbalance and unequal distribution of instances among clients. To better embody these circumstances, we've generated a new representative network intrusion dataset for FL-enabled IDSes.

V. Conclusions

In this paper, we introduced a novel benchmark dataset, *FLNET2023* for FL-based network intrusion detection systems. We generated the dataset using the CORE emulator featuring a realistic network topology and a diverse range of traffic and attack types. We thoroughly analyzed the dataset, establishing its unique characteristics, and assessed the efficacy of FL aggregation algorithms in the context of IDSes. Our experimental results highlight the challenges posed by local class imbalance indicating a clear path for future research to enhance the efficacy of FL-based IDS. Our work can serve as a foundation for further explorations into network intrusion detection, while the newly created dataset will provide a realistic basis for such investigations. In the future, we plan to solve the challenges highlighted by our dataset.

Acknowledgment

This work was sponsored by the DEVCOM Analysis Center and was accomplished under Cooperative Agreement Number W911NF-22-2-0001. This work was also partially funded by the U.S. Department of Energy, DE-SC0023392.

References

[1] B. McMahan, E. Moore, D. Ramage, and et al., "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[2] N. A. A. Al-Marri, B. S. Ciftler, and M. M. Abdallah, "Federated mimic learning for privacy preserving intrusion detection," in *IEEE BlackSeaCom*. IEEE, 2020, pp. 1–6.

[3] S. A. Rahman, H. Tout, C. Talhi, and A. Mourad, "Internet of things intrusion detection: Centralized, on-device, or federated learning?" *IEEE Network*, vol. 34, no. 6, pp. 310–317, 2020.

[4] J. Zhang, P. Yu, L. Qi, S. Liu, H. Zhang, and J. Zhang, "FLDDoS: DDoS attack detection model based on federated learning," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2021, pp. 635–642.

[5] M. Tavallaee, E. Bagheri, W. Lu, and et al., "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on CISDA*. IEEE, 2009, pp. 1–6.

[6] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *MilCIS Conference*. IEEE, 2015, pp. 1–6.

[7] 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.

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

[9] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "CORE: A real-time network emulator," in *MILCOM*. IEEE, 2008, pp. 1–7.

[10] J. Li, L. Lyu, X. Liu, and et al., "FLEAM: A federated learning empowered architecture to mitigate ddos in industrial IoT," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 6, pp. 4059–4068, 2021.

[11] Q. Qin, K. Poularakis, K. K. Leung, and L. Tassiulas, "Line-speed and scalable intrusion detection at the network edge via federated learning," in *IFIP Networking Conference*. IEEE, 2020, pp. 352–360.

[12] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization." *ICISSP*, vol. 1, pp. 108–116, 2018.

[13] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[14] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, and N. D. Lane, "Flower: A friendly federated learning research framework," *arXiv preprint arXiv:2007.14390*, 2020.

[15] T.-M. H. Hsu, H. Qi, and M. Brown, "Measuring the effects of non-identical data distribution for federated visual classification," *arXiv preprint arXiv:1909.06335*, 2019.