Federated Learning-Based In-Network Traffic Analysis on IoT Edge

Mingyuan Zang*, Changgang Zheng†, Tomasz Koziak‡, Noa Zilberman†, and Lars Dittmann*
*Technical University of Denmark, ‡Netlight, †University of Oxford
Email: *{minza, ladit}@dtu.dk, ‡tomasz.koziak@netlight.com, †{name.surname}@eng.ox.ac.uk

***Abstract*—The rise of IoT-connected devices has led to an increase in collected data for service and traffic analysis, but also to emerging threats and attacks. In-network machine learning-based attack detection has proven effective in fast response, but scaling to distributed IoT edge devices risks increasing communication overheads and raising data privacy concerns. To address these concerns, we present FLIP4, a distributed in-network attack detection framework based on federated tree models. FLIP4 maintains data privacy by enabling distributed machine learning training while keeping data local on IoT edge, and provides in-network inference within the programmable data plane on edge gateway for timely attack labeling and mitigation. Evaluation results show that FLIP4 can accurately detect attacks while maintaining source data privacy and enabling lightweight deployment on IoT edge.**

***Index Terms*—In-Network Computing; Federated Learning; Security; Internet of Things; P4**

I. Introduction

As more devices from multiple vendors are connected to the Internet of Things (IoT) network, an increasing volume of data is collected from end devices. To efficiently process the data and obtain useful information to make intelligent decisions on network management and performance optimizations, machine learning (ML) algorithms have been deployed for traffic analysis. For instance, ML algorithms have been studied for traffic engineering or attack detection [1, 2].

To provide accurate analysis capability, a large amount of telemetry data is sent to server/cloud for ML training and inference. Such transmissions increase the load on the communications infrastructure and may lead to long Round-Trip Times (RTT), increasing the response time to mitigate attacks. With 6G's extremely low latency requirements, it is vital to detect and mitigate malicious traffic timely to prevent further damage to IoT networks. Offloading attack detection services from the remote server/cloud to the network edge has been proposed as a means to provide prompt reaction [3, 4]. The advent of programmable data planes and in-network ML inference have further driven possibilities for in-network attack detection [5]. Building upon programmable pipelines and P4 language, an ML inference process can be mapped to the data plane to directly make a detection decision along with packet forwarding. By flexibly parsing packets' features and applying ML-based processing, potential threats can be labeled and mitigated immediately without affecting other parts of the network.

Although such in-network ML-based attack detection offers immediacy and flexibility, scaling such solutions to multiple devices in a distributed system poses challenges, particularly in the case of multiple gateway access at the IoT edge [6]. To obtain a global analysis of the devices, it is necessary to share the collected data or statistics with the controller/server to build the traffic profile. The aggregated profile can be used for global ML model training and information sharing among the gateway devices for local model updates. However, building the profile and coordinating the learning updates in an efficient and lightweight manner for IoT scenario presents challenges [7]. 1) Contributing the collected data to controller/server can result in an increase in communication overhead as the system scales to multiple gateways. 2) The collection and sharing of traffic data may raise privacy concerns and may require compliance with data protection regulations. Addressing these challenges requires efficient and lightweight techniques for integrating the traffic profile and coordinating the learning process in a secure manner.

To tackle these issues, Federated Learning (FL) algorithms have been proposed [8] to enable ML-based model training and inference with the data maintained locally. FL enables distributed ML processes in a federated manner. Data volume and communication overheads are reduced by keeping the input data locally, while sharing model parameters with the server to maintain data privacy. FL in previous work has been mainly applied to images and texts, and deep learning algorithms are used to learn such data inputs [8]. In IoT scenarios, tabular data like short messages or packet fields are the common information parsed from the collected IoT traffic. To analyze such data, deep learning algorithms may require complex neuron-based training with high overheads. Conversely, tree-based learning models show advantages in accurate performance and easy deployment [2, 9]. To drive the ensemble tree-based model to play its advantages in FL-based attack detection service on IoT edge, there is still a lack of discussion and solutions to efficiently coordinate the tree-based FL design with the in-network ML-based attack detection in distributed IoT scenario.

In this work, we present FLIP4, a distributed in-network attack detection solution based on federated tree-based models. FLIP4 does model inference in-band to quickly detect and

This work was partly funded by VMWare, EU Horizon SMARTEDGE (101092908), and the Nordic University Hub on Industrial IoT (HI2OT) by NordForsk.

mitigate attacks. It utilizes FL to coordinate an efficient distributed deployment of in-network attack detection, reducing communication overheads and preserving data privacy. Instead of sharing all collected data, FLIP4 trains a global model by sharing local model weights without leaking any source data.

The main contribution of this work is as follows:

- We propose FLIP4, a distributed privacy-preserving framework for in-network attack detection and mitigation on IoT edge. (§III)
- FLIP4 introduces FL and Differential Privacy within the framework to coordinate the model learning and inference process, to reduce communication overheads, and to enhance data privacy. (§IV-B)
- FLIP4 coordinates in-network inference and model updates at runtime to achieve model mapping and update processes in a flexible and automated manner. (§IV-A)

We present an evaluation showing that FLIP4 achieves accurate attack detection while maintaining source data privacy. Additionally, the framework is lightweight and can be easily deployed on IoT edge with negligible overhead. (§V)

II. Related Work

A. Traffic Analysis on IoT Edge

The heterogeneity and dynamic distributed deployment of IoT end devices like sensors and actuators have left vulnerabilities and attack vectors for malicious activities. Meanwhile, IoT devices commonly have limited computing resources and may operate on low-power configurations, resulting in challenges in efficient security mechanisms. Traffic analysis has been studied to learn the traffic pattern and detect the potential risks in IoT networks [10]. To identify malicious traffic from the dynamic network, ML has been applied to detect anomalies and deviations. The application of ML can provide competent attack detection over the distributed deployment of IoT devices [11].

B. Programmable Data Plane

Programmable data plane allows packet processing customization on network devices. In a programmable data plane, custom packet processing logic can be defined using a programming language such as P4 [12]. P4 can define match-action (M/A) tables and pipelines of the programmable data plane. As a lookup table, M/A table is used to match packets to specific actions based on their headers. When a packet arrives, its header is parsed for looking up an entry in the match-action table. The corresponding action associated with that entry is then executed to determine how the packet should be processed. Such programmability and flexibility have driven various research fields like in-network ML.

C. In-network Machine Learning

In-network machine learning (ML) techniques provide line rate inference on programmable network devices, within the network. Different from traditional ML services that train and deploy models either on a server or an accelerator (e.g., GPU), in-network ML first trains a model on a server (control plane), then converts the model to a series of packet processing logics or M/A table rules, and finally loads the model to do inference on a network device (data plane). Previous works have covered ML models such as tree models (Decision Tree, Random Forest, XGBoost (XGB)) [4, 13, 14, 15], Neural Network (NN) [6, 16, 17], and other classical models (k-means, Support Vector Machine, and Naïve Bayes) [1, 3, 18]. The benefits of running ML models on the data plane are low-latency response and line-rate processing ability. However, due to hardware limitations of programmable network devices, mapped models trade-off accuracy and model size, and have different update flexibility.

D. Federated Learning

Federated Learning (FL) provides a distributed learning scheme to keep the data locally and avoid directly sending the data to the server. Classical ML algorithms can be trained locally in a distributed manner and the parameters of the models are sent to the server for averaging and obtaining the global model. Existing work has applied FL for anomaly detection in network traffic to label potentially malicious behaviors. For instance, researchers in [7] deployed a NN for local training to enable automated ML-based attack detection in IoT scenarios. Differential Privacy (DP) has been studied to further enhance training privacy [19], using techniques such as clipping and adding random noise during the gradient computation process. DP uses those to increase the difficulty of inverting parameter data of shared models and mitigate the risk of attackers intercepting private information.

E. Design Challenges

Despite the potential benefits of deploying in-network ML and FL separately, some challenges need to be addressed to deploy the FL together with in-network ML for accurate and timely attack detection over the distributed gateways on IoT edge. Besides the challenges listed in the introduction section, more challenges need to be considered with respect to the FL deployment: 1) How to enable lightweight learning and update process to adapt to distributed IoT setup. 2) How to secure the model-sharing process with low overhead to prevent malicious interception. 3) How to integrate the local information into a global model and drive the local update for in-network inference process. In this work, FLIP4 is proposed to approach these challenges by introducing federated tree models with DP to in-network ML inference process and providing timely attack detection and mitigation service on IoT edge.

III. Proposed Design

Network Scenario. In this work, we consider a network scenario with multiple switches acting as IoT gateways that are deployed between the IoT end devices and a remote server in the cloud. Each IoT end device is assumed to be at risk of network attacks like Scanning, Man-in-the-Middle (MitM), Distributed Denial of Service (DDoS), etc. An attacker might exploit one of the devices to launch protocol attacks or use several of them as a botnet.

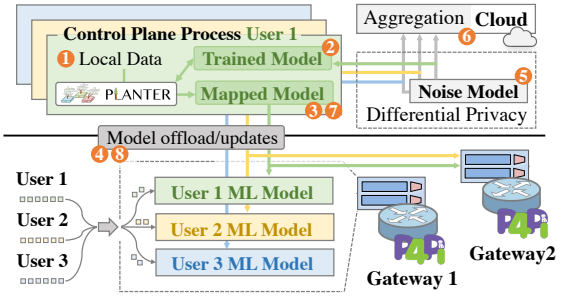

Fig. 1. System design of FLIP4.

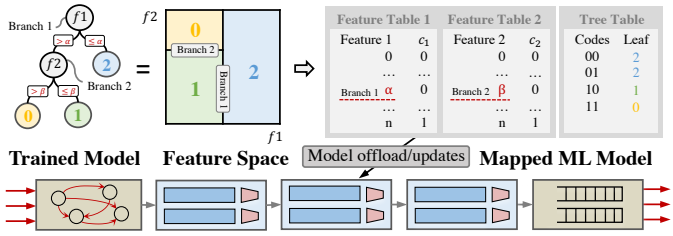

Fig. 2. ML model inference mapping process to a programmable data plane in an encode-based method. A single decision tree is depicted as an example.

System Design. The proposed design is a privacy-preserving in-network analysis framework based on FL and in-network ML. Under this framework, traffic is being processed inside the IoT gateway with low latency and high throughput, traffic data remain locally, and uploaded models are added with noise to prevent privacy leakage. The complete process consists of the following steps. A gateway (a switch) is first initialized with a local model trained using a local traffic dataset (Figure 1 step ❶ and ❷). The trained model is offloaded and mapped to the data plane pipeline for in-network inference by running a P4 code and inserting mapped table entries reflecting the model structure and parameters (Figure 1 step ❸ and ❹). Incoming traffic from each user device passes through the programmable data plane on IoT gateway for header parsing and feature extraction. The extracted information goes through the M/A pipeline with inference logic for labeling. If a packet is labeled as benign, it will be forwarded, otherwise, it would be dropped. To maintain a global model for the server over the local models on multiple gateways, the parameters of the trained model on each IoT gateway are sent to the server with the protection of Differential Privacy (as Figure 1 step ❺). When the server receives the updated information from all gateways, the aggregator starts to average the parameters to generate a global model, and the parameters of this global model are sent back to each IoT gateway for local model updates (as Figure 1 step ❻). The mapper on each gateway coordinates the control plane and data plane to map the updated model's parameters to table rules and insert the new rules to the data plane pipeline at runtime (as Figure 1 step ❼ and ❽). The training process is further split into model training and communication process. In this system workflow, the model training includes local training on the IoT gateway, and global training on the server. Communication refers to the interaction between the control plane and data plane within the IoT gateway, and to the parameter exchange between the IoT gateway and the server.

The FLIP4 system combines interaction between the data plane, control plane, and server to achieve FL-based in-network attack detection. It combines three types of components: trainer, mapper, and aggregator.

- Trainer: The role of the trainer is to train the local model on every single device. It runs on the IoT gateway control plane and preprocesses local data. It next initializes the local model and conducts the model training.
- Mapper: The completeness of model training will trigger the mapper to map the model inference process to the data plane. It maps the trained model to P4 code and a set of M/A table rules [4]. These rules are inserted by the control plane into the data plane pipeline for in-network inference. The data plane of each IoT gateway runs the generated P4 code, and M/A rules are written to the data plane's tables for in-network inference at runtime.
- Aggregator: When the trainer completes model training based on the local data, the parameters of the local model are sent to the server, where the aggregator computes a federate averaging over all local models to gain the global model. It then sends the averaged parameters back to the IoT gateway for local updates.

Figure 1 illustrates a scenario where multiple users (end devices) access through the same gateway in a multiple access gateway scenario. This includes a more dedicated scenario when a gateway serves the data access from a single user. Depending on the deployment location and service requirement, the access scenario may vary but can both be supported in FLIP4 with the same process for FL-based in-network attack detection and mitigation.

IV. Implementation

A. In-network Attack Detection and Model Mapping

To enable attack labeling and mitigation, a trained ML model is mapped to data plane as a part of ingress processing using P4 code and a set of M/A rules. Several models have been studied in related work for mapping, which includes tree-based models and neural networks [6]. In this work, FLIP4 uses tree-based ensemble models like XGB [13]. Compared with NN [7], tree-based models perform better, accuracy wise, on tabular data which fits the packet features. Due to the "simpler" structure and similarity to the M/A pipeline, tree-based models utilize fewer resources than the NN [13].

Data Preprocessing. To achieve in-network attack detection and mitigation, features need to be extracted and prepared for ML inference before a packet is forwarded to the next hop. By defining the packet header parser and pipeline processing in P4 language, features can be extracted to metadata within the data plane. The extracted features are preprocessed into a format suitable for ML inference. They are then used for in-network ML inference to do the classification task and decide whether the packet is benign or malicious. In this work, features like port information and TCP-related flags [*protocol, src port, dest port, SYN flag, ACK flag*] are extracted and processed for model training and inference.

In a single-switch deployment in previous work [5], it is assumed that collected traffic features are shared in digests to the control plane/server. In FLIP4, such data sharing step is avoided by introducing the FL method to share model parameters instead. In this process, data distribution could be a concern as it may affect the local model and the model integration performance at the server might be affected [20]. For instance, it is usually assumed that the data is independent and identically distributed (IID) in a centralized ML model. This might not always be the case in IoT edge, considering that the local data is collected across different devices. The distribution of the data may vary if the end devices have different network configurations or are at risk of different types of vulnerabilities or attacks. Thereby, both IID and non-IID scenarios are studied in this work.

Model Mapping and Update. When a model completes initial local training, the model structure is mapped to the data plane for in-network inference. This involves breaking down the model into a series of simple operations that can be performed by the processing pipeline. Figure 2 depicts a simplified pipeline and demonstrates this process. When a model finishes its training process, a model structure consisting of tree nodes and branches is obtained as plotted in the upper left corner of Figure 2. Such a tree structure gives a splitting example of feature space with two features (as the square plot on the right side of "="). For each split (decision boundary/branch) in the tree, feature space is divided into two regions with one region indicating the "left" branch and the other region indicating the "right" branch. When a data sample traverses the tree, the sequence of branches that a data sample follows forms a feature split path, where the value of input features are compared against the threshold values, and then proceeds down the appropriate branch of the tree. To map such model structure to the data plane pipeline, M/A table rules are generated by translating the feature split path into the feature table and tree table. The encode-based solution of Planter [13] is used to encode the parameters. The feature tables use input feature values and feature splits as match conditions and the encoded values as actions pointing to the next tree node and branch. The tree table records the code pairs as match conditions to output the labeling decisions.

With this mapping workflow, model updates can be enabled in a more flexible manner. Compared with methods that hard-code the model parameters in P4 code, mapping the model to table entries provides the flexibility to change and update the model at runtime building upon the architectural advantages of programmable data plane.

B. FLIP4's Federated Learning and Distributed Deployment

Turning an in-network ML inference design into a distributed one with FL on multiple gateways, requires addressing several problems.

Local Results Integration. In previous work, *FedAvg* algorithm [21] was used to update the weights of neurons in NN model. In tree-based ensemble models, the results' integration method is adapted to the tree structure. To update the tree

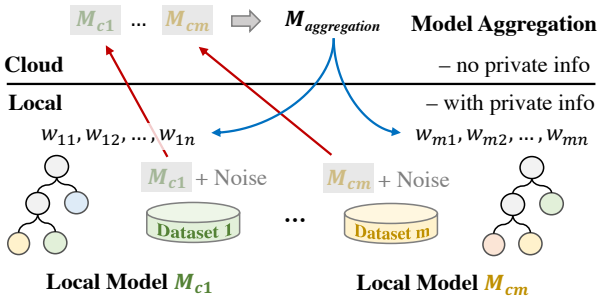

Fig. 3. Sample workflow of FL function for model sharing and parameter updates with DP.

model with a set of data samples $D = \{(x_i, y_i)\}_{i=1}^{n}$, the objective function of tree f can be denoted as:

$$\mathcal{L}(f) \approx \sum_{i=1}^{n} \left(\ell\left(y_i, \hat{y}\right) + g_i f\left(x_i\right)\right) + \Omega \tag{1}$$

where l is leaf node, $\hat{y}$ is prediction of x, g_i is a gradient of the loss function, and $\Omega = \gamma L + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$ (γ, λ are penalises). To update model parameters means to update the gradient weight $w_l = -\frac{\sum_{i \in I_l} g_i}{|I_l| + \lambda}$ [9]. The weights of leaf nodes contribute to the vote and affect the predicted value. If the leaf weight is higher, the corresponding data sample is more likely to be classified as positive, and vice versa. Model updates therefore aim to update such weights or probabilities in leaf nodes.

Model Update. Figure 3 depicts a sample workflow of the FL function in FLIP4. To start with, the server initializes a model structure and sends it out to M local devices $[D_1, ..., D_m]$. With this information to initialize the model structure, the local model is trained using the local data. The parameters of the trained local model $[w_{11}, ..., w_{1n}]$ are packed and sent from each device to the server. Extra noise is added for DP to protect the model-sharing process. To update the global model, the central server aggregates the local models from each device $[M_{c1}, ..., M_{cm}]$ using a weighted average. The aggregated model is then sent back to the local devices. To update the local model, each device takes the averaged weights computed as previously described to update the decision boundaries of the tree, as well as the corresponding newly generated M/A table entries to map the updated inference.

Node Communications. FL can help reduce communications cost, compared with a scenario where all data is forwarded to the cloud. By properly selecting the communication mechanism, the communications cost can be further optimized. In the IoT scenario, devices require a communication protocol that can support low-power devices, such as web sockets. A socket has a low delay when sending models between parties [22]. To keep FLIP4's gateways synchronized in model's update process, a time window is set at the server, ensuring that the averaging process is only triggered when all parameters are received from gateway nodes. Thereby, model updates can be ensured in a lightweight and reliable way.

As a transmission of model's parameters may be intercepted by the attackers and leak the model information, Differential Privacy (DP) is applied to the parameter communication process by adding random noise to the model information. DP can

TABLE I
DETECTION ACCURACY WITH/WITHOUT DP ON CICIDS 2017 DATASET.

		Gateway1	Gateway2	Gateway3	Global	Offline
NN (Baseline)	IID	1.0	1.0	1.0	1.0	1.0
	Non-IID	0.9982	0.8430	0.5125	0.9649	0.9999
XGB (FLIP4)	IID	1.0	1.0	1.0	1.0	1.0
	Non-IID	0.9994	0.8124	0.5836	0.9681	0.9999
DP-XGB (FLIP4)	IID	0.9182	0.8991	0.9467	0.9417	1.0
	Non-IID	1.0	0.9064	0.6367	0.9274	0.9999

measure and control the leakage of the model information by statistically characterizing the impact of a single data sample on the model. The concept *privacy budget* [9] is used to evaluate the performance of the differential privacy mechanism and it is expected to be as small as possible. Nonetheless, adding noise will lead to a performance loss in the model. Thus, there is a trade-off between privacy and model performance. Rényi Differential Privacy [19] with Gaussian mechanism [9] is used in this work by adding calibrated Gaussian noise to keep the simplicity and succinctness.

V. EVALUATION AND EXPERIMENTAL RESULTS

Experimental Setup FLIP4's data plane code is implemented in P4 language using bmv2 with v1model architecture and prototyping on Raspberry Pi using P4Pi-v.0.0.3 [23]. Python code provides controller and server functionality. FLIP4 extends the design in FL algorithm [9] and *Planter* [13] for ML training and inference functions. To evaluate FLIP4's performance on multiple nodes, Mininet is used for network emulation. Baseline results are taken from offline model learning using NN in *PyTorch* and XGB in *sklearn*.

Network Setup: As FLIP4 is deployed for a distributed network scenario, a SOHO (Small Office/Home Office) network scenario is taken as a sample scenario where a limited number of gateways are connected. A network topology is built in Mininet to simulate a connection at the network edge with multiple switches acting as gateways and these gateways are linked with an edge switch connecting to the server. The parameters of the topology vary between the experiments.

Dataset: Public dataset CICIDS 2017 [24] is used as source data, which includes both benign and malicious network traffic. In this work, we use Scanning, DDoS, and Botnet attacks as emerging attacks from IoT end devices assuming that remote server/cloud has to learn these attacks.

A. Evaluation Metrics

Detection Accuracy: Several metrics can be used to evaluate detection performance. In this paper, we use Accuracy (ACC), True Positive Rate (TPR)/ False Positive Rate (FPR), Area Under the ROC Curve (AUC) as the metrics to evaluate detection performance. They are defined as below, where $TPR = \frac{TP}{TP+FN}$ and $FPR = \frac{FP}{TN+FP}$.

$$ACC = \frac{TP+TN}{TP+TN+FP+FN}, AUC = \int TPR\, d(FPR) \quad (2)$$

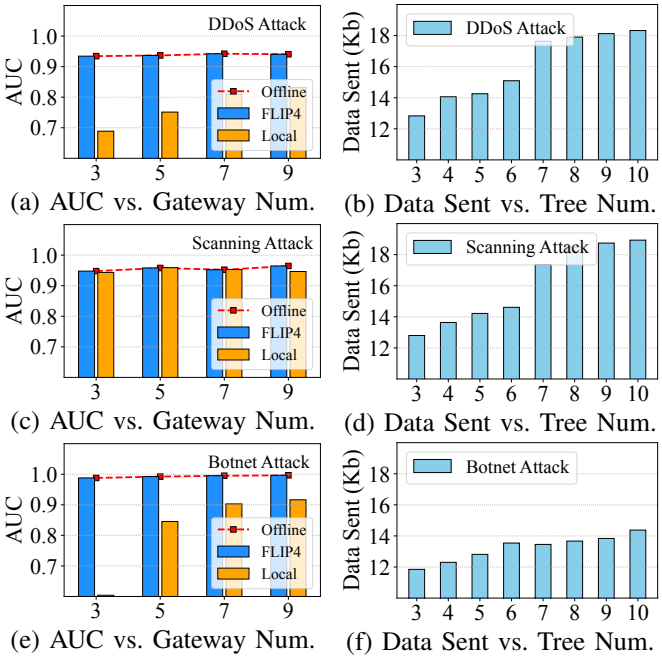

Fig. 4. Scalability evaluation of FLIP4 (DP-XGB). Offline - model trained by source data, FLIP4 - global model, local - local model. Graphs (a),(c),(e) present detection accuracy in terms of AUC score vs. number of switches under different attacks. Graphs (b),(d),(f) present data sent vs. number of switches. The attacks shown are (a),(b) DDoS, (c),(d) Scanning, (e),(f) Botnet.

Communication Overhead: To evaluate the volume of data being sent to the server for model sharing, the number of sent bytes is recorded as communication overhead.

B. Experimental Results

Detection Accuracy. Table I provides a summary of detection accuracy of FLIP4 using *CICIDS 2017* dataset. Taking the XGB model in FLIP4 as an example, Table I presents the detection performance of the model implemented in an in-network manner with/without DP (DP-XGB/XGB) enabled on IID/non-IID dataset, where IID data is obtained by shuffling the dataset. To clearly list the performance results of local models at gateways and the global model in server, a network topology with 3 access gateways is setup on edge. Learning results on source data at the server are also listed as offline results. The results show that: a) Based on local information and local training, the global model is able to integrate the local models and present accurate attack detection without directly gathering the local data at the server. Due to the privacy trade-off, FLIP4's global accuracy is slightly lower than offline results that are directly trained from source data, but is higher than NN model's global accuracy. b) FLIP4 provides higher detection accuracy for IID data than for non-IID data. Considering that the global model is an average of the model at each gateway (switch), the distribution of local data may affect the model averaging performance. c) Introducing DP into the FL model can lead to a minor accuracy degradation (e.g., 4% decrement in accuracy for non-IID data). Such degradation is caused by the introduction of noise in DP method, which is a trade-off between accuracy and privacy.

Scalability. As FLIP4 is designed to achieve distributed deployment on multiple gateways (switches), scalability tests are done to evaluate how FLIP4 can scale to a different number of gateways. Figure 4 (a) (c) (e) show the detection accuracy results in terms of AUC on three types of attacks (a-DDoS, c-Scanning, e-Botnet) when the XGB model is trained with 5 trees and a depth of 6. The bars present a general trend that detection accuracy slightly increases as more gateways are added to the network and promoted to the local training. This is especially the case for Botnet attacks where the local gateways suffer from botnet exploitation traffic.

Communication Overhead. Figure 4 (b) (d) (f) illustrate how the volume of data may vary as the model size scales up under different attack patterns. In this figure, as the number of trees in the model increases, the model size also increases, resulting in a larger number of parameters that are shared with the server for global integration during the decision-making process. The average volume of data sent from each local gateway to the server also increases. This data volume is similar to Logistic Regression-based FL [25] and is less than the one based on NN [20] . Note that this figure presents the effect of a number of trees in communication overhead, other parameters like the depth of trees may also cause effects. For instance, the data volume jumps from 6 trees to 7 trees in Figure 4 (b) (d) is due to the increasing depth in the trained model. When it comes to the processing time in this setup, it takes $\sim$0.011s for the local gateway to do training and $\sim$0.712s for the server to do global integration and update. For in-network ML inference, it takes $\sim$0.107s for the controller to insert table entries. In this experiment, time window for global integration is set to 15s when 5 local gateways are connected, where the window setup may vary by considering the number of gateways, as the integration in FLIP4 is triggered only when all gateways complete the model sharing to ensure reliability. Asynchronous FL [26] can provide a more flexible integration process by reducing the waiting time for local sharing.

VI. Conclusion

In this paper, we present FLIP4, an FL-based in-network attack detection and mitigation solution for distributed gateway deployment. In-band packet processing and in-network ML inference are configured for timely attack detection and mitigation on IoT edge. FL is introduced to adapt to the distributed setup, connecting multiple edge gateways. With FL introduced, the server can train a global model over the local model received from each IoT gateway without directly querying traffic data. In this manner, privacy can be preserved and communication overheads can be reduced. Differential Privacy is used to improve the privacy of the model-sharing process. Results show that FLIP4 can achieve accurate in-network attack detection and obtain an accurate global model. The framework deployment achieves flexible design with in-network ML inference providing low-latency attack detection and mitigation on IoT edge as well as FL preserving the data privacy and model updates with insignificant communication overhead. *Future work* may extend the design by focusing on asynchronous FL integration and deployment on heterogeneous software or hardware devices.

References

[1] C. Zheng, Z. Xiong, T. T. Bui, S. Kaupmees, R. Bensoussane, A. Bernabeu, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "IIsy: Practical In-Network Classification," 2022.

[2] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 29–35.

[3] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM workshop on hot topics in networks*, 2019, pp. 25–33.

[4] C. Zheng and N. Zilberman, "Planter: seeding trees within switches," in *SIGCOMM'21 Poster and Demo Sessions*, 2021, pp. 12–14.

[5] M. Zang, C. Zheng, R. Stoyanov, L. Dittmann, and N. Zilberman, "P4pir: in-network analysis for smart iot gateways," in *Proceedings of the SIGCOMM'22 Poster and Demo Sessions*, 2022, pp. 46–48.

[6] 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*. IEEE, 2020, pp. 352–360.

[7] 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," *ICDCS*, pp. 756–767, 2018.

[8] J. Konečnỳ, B. McMahan, and D. Ramage, "Federated optimization: Distributed optimization beyond the datacenter," *arXiv preprint*, 2015.

[9] S. Maddock, G. Cormode, T. Wang, C. Maple, and S. Jha, "Federated boosted decision trees with differential privacy," in *CCS '22*, 2022.

[10] N. Hadar, S. Siboni, and Y. Elovici, "A lightweight vulnerability mitigation framework for iot devices," in *IoTS&P '17*. New York, NY, USA: Association for Computing Machinery, 2017, p. 71–75.

[11] R. Kozik, M. Choraś, M. Ficco, and F. Palmieri, "A scalable distributed machine learning approach for attack detection in edge computing environments," *JPDC*, vol. 119, pp. 18–26, 2018.

[12] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, and et al., "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, jul 2014.

[13] C. Zheng, M. Zang, X. Hong, R. Bensoussane, S. Vargaftik, Y. Ben-Itzhak, and N. Zilberman, "Automating In-Network Machine Learning," 2022. [Online]. Available: https://arxiv.org/abs/2205.08824

[14] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pforest: In-network inference with random forests," 2019.

[15] X. Hong, C. Zheng, S. Zohren, and N. Zilberman, "Linnet: Limit order books within switches," in *SIGCOMM'22 Poster*, 2022.

[16] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," in *NSDI 22*, 2022, pp. 513–533.

[17] T. Swamy, A. Rucker, M. Shahbaz, and K. Olukotun, "Taurus: An intelligent data plane," *arXiv preprint arXiv:2002.08987*, 2020.

[18] R. Friedman, O. Goaz, and O. Rottenstreich, "Clustreams: Data plane clustering," in *SOSR*, 2021, pp. 101–107.

[19] I. Mironov, "Rényi differential privacy," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017, pp. 263–275.

[20] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, and M. Jirstrand, "A performance evaluation of federated learning algorithms," ser. DIDL '18, New York, NY, USA, 2018, p. 1–8.

[21] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2016.

[22] L. Heiko, B. Nathalie, T. Gegi, Z. Yi, and et al., "IBM federated learning: an enterprise framework white paper V0.1," *CoRR*, vol. abs/2007.10987, 2020. [Online]. Available: https://arxiv.org/abs/2007.10987

[23] R. Stoyanov, A. Wolnikowski, R. Soulé, S. Laki, and N. Zilberman, "Building an internet router with P4Pi," in *ANCS*, 2022, p. 151–156.

[24] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *ICISSP 18*, 2018, pp. 108–116.

[25] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-effective federated learning design," in *IEEE INFOCOM 21*, 2021, pp. 1–10.

[26] J. Liu, H. Xu, L. Wang, Y. Xu, C. Qian, J. Huang, and H. Huang, "Adaptive asynchronous federated learning in resource-constrained edge computing," *IEEE TMC*, vol. 22, no. 2, pp. 674–690, 2023.