Federated In-Network Machine Learning for Privacy-Preserving IoT Traffic Analysis

Zang, Mingyuan; Zheng, Changgang; Koziak, Tomasz; Zilberman, Noa; Dittmann, Lars

Published in:
ACM Transactions on Internet Technology

Link to article, DOI:
10.1145/3696354

Publication date:
2024

Document Version
Publisher's PDF, also known as Version of record

Federated In-Network Machine Learning for Privacy-Preserving IoT Traffic Analysis

MINGYUAN ZANG, DTU Electro, Technical University of Denmark, Kgs Lyngby, Denmark
CHANGGANG ZHENG, University of Oxford, Oxford, United Kingdom of Great Britain and Northern Ireland
TOMASZ KOZIAK, Netlight, Copenhagen, Denmark
NOA ZILBERMAN, University of Oxford, Oxford, United Kingdom of Great Britain and Northern Ireland
LARS DITTMANN, Technical University of Denmark, Lyngby, Denmark

The expanding use of Internet-of-Things (IoT) has driven machine learning (ML)-based traffic analysis. 5G networks' standards, requiring low-latency communications for time-critical services, pose new challenges to traffic analysis. They necessitate fast analysis and response, preventing service disruption or security impact on network infrastructure. Distributed intelligence on IoT edge has been studied to analyze traffic, but introduces delays and raises privacy concerns. Federated learning can address privacy concerns, but does not meet latency requirements. In this article, we propose FLIP4: an efficient federated learning-based framework for in-network traffic analysis. Our solution introduces a lightweight federated tree-based model, offloaded and running within network devices. FLIP4 consumes less resources than previous solutions and reduces communication overheads, making it well-suited for IoT edge traffic analysis. It ensures prompt mitigation and minimal impact on services in the presence of false alerts using two approaches (metering and dropping), thereby balancing learning accuracy and privacy requirements.

CCS Concepts: • **Networks** → **In-network processing**; **Network security**; • **Computing methodologies** → **Machine learning algorithms;**

Additional Key Words and Phrases: In-network computing, federated learning, security, internet of things, P4

ACM Reference Format:
Mingyuan Zang, Changgang Zheng, Tomasz Koziak, Noa Zilberman, and Lars Dittmann. 2024. Federated In-Network Machine Learning for Privacy-Preserving IoT Traffic Analysis. *ACM Trans. Internet Technol.* 24, 4, Article 29 (November 2024), 24 pages. https://doi.org/10.1145/3696354

This work was partly funded by the Nordic University Hub on Industrial IoT (HI2OT) by NordForsk, VMware, and Innovate UK (Project No. 10056403) as part of the SmartEdge EU project (Grant Agreement No. 101092908).
Authors' Contact Information: Mingyuan Zang, DTU Electro, Technical University of Denmark, Kgs Lyngby, Denmark; e-mail: minza@dtu.dk; Changgang Zheng, University of Oxford, Oxford, United Kingdom of Great Britain and Northern Ireland; e-mail: changgang.zheng@eng.ox.ac.uk; Tomasz Koziak, Netlight, Copenhagen, Denmark; e-mail: tomasz.koziak@netlight.com; Noa Zilberman, University of Oxford, Oxford, United Kingdom of Great Britain and Northern Ireland; e-mail: noa.zilberman@eng.ox.ac.uk; Lars Dittmann, Technical University of Denmark, Lyngby, Hovedstaden, Denmark; e-mail: ladit@dtu.dk.

ACM 1533-5399/2024/11-ART29
https://doi.org/10.1145/3696354

1 Introduction

Distributed intelligence has been increasingly recognized for its scalability and flexibility compared to centralized intelligence, especially in the context of **Internet-of-Things (IoT)** networks. Studies have highlighted its effectiveness in providing optimal solutions for collective information and holistic views in IoT networks [33, 42]. It has been applied for traffic analysis services in IoT networks, such as device identification [13] and anomaly detection [55]. However, existing distributed intelligence-based solutions [6, 52] primarily focus on accurate analysis and decisions. These solutions, while effective, fall short in fast response and mitigation following decision-making, as well as in ensuring efficient communication among distributed nodes.

This limitation becomes particularly critical as networks evolve to support **ultra-reliable and low-latency communication (URLLC)** [10]. In such environments, the ability to quickly respond to incidents is crucial, given that emerging attacks can significantly impact network infrastructure if not promptly addressed. This risk is notably high in IoT networks, where end devices often lack security measures due to resource constraints and performance considerations [48]. Consequently, there is a pressing need for active defense services to detect and mitigate attacks effectively and quickly [8].

Federated Learning (FL) has provided a solution to collaboratively train and improve distributed intelligent models over time. It introduces distributed machine learning-based deployment across multiple distributed nodes while keeping the data localized. Despite the demonstrated efficiency and privacy in traffic analysis, FL has design challenges in high communication overhead and unstable connection of local nodes [57]. This affects the performance of the FL-based traffic analysis in IoT networks where network edge devices are relatively dynamic and have limited computing resources.

Minimizing the action enforcement time for identified anomalies remains another challenge of applying FL-based traffic analysis for time-critical services. The emerging attacks have necessitated fast mitigation responses, as recent reports indicate a significant increase in DDoS and Botnet attacks in IoT networks, exploiting protocol vulnerabilities in IoT devices [2]. Prior work [11, 33] has explored FL to enable accurate traffic analysis but falls short in fast response and action enforcement. Providing **Machine Learning (ML)**-based analysis at line-rate within IoT edge network devices, as traffic is forwarded through the devices, can potentially ensure quick reactions to identified traffic issues. This opportunity has been made possible by recent progress in in-network ML inference [63, 65]. It offloads ML inference to the data plane within network devices (e.g., programmable switches and SmartNICs), rather than depending solely on general processing resources (e.g., CPU/GPU) [60].

State-of-the-Art (SOTA) work [40] employed in-network traffic analysis on IoT edge using Neural Network-based FL preserving privacy. This work incorporated **Binary Neural Networks (BNN)** and quantization. While its approach achieved a privacy-preserving traffic analysis on IoT edge, quantization compromised accuracy. It focused on detection, without a solution for mitigation. Moreover, the BNN introduced processing overheads and latency to normal traffic. The BNN model itself, demonstrated on a smartNIC [40], is not well-suited for implementation on resource-constrained programmable network devices [64].

Combining the opportunities in in-network traffic analysis and existing limitations in FL, we ask this question: *Can we realize a practical in-network FL framework for resource-constrained IoT edge?* Specifically, a framework enabling high accuracy, fast response, and privacy-preserving in-network traffic analysis at high performance and low overhead.

To answer this question, we propose FLIP4, an efficient ML-based traffic analysis framework within distributed IoT gateway devices that achieves (1) accurate traffic analysis with low-latency response to mitigate issues, (2) privacy-preserving traffic knowledge sharing, and (3) low

deployment overhead. We address the aforementioned challenges in fast analysis and response, privacy-preserving information sharing, and lightweight deployment. To allow lightweight knowledge sharing for distributed intelligence, we incorporate tree-based model with FL-based model training, which consumes less local resources and lower communication overheads than typical NN-based models. To enable accurate traffic analysis with fast response to detected anomalies, we deploy the tree-based model in a novel in-network manner within local resource-constrained IoT gateways. When it comes to mitigation measures, we introduce two sample options, dropping and metering, as enforced actions when the model identifies issues. While direct dropping is suitable for prompt mitigation, metering reduces the impact on a running service when the model gives false alerts. The in-network model is reconfigured at runtime to adapt to traffic dynamism without disrupting network traffic.

In summary, our contributions are:

— We propose a FL-based framework within distributed IoT gateways for privacy-preserving traffic analysis at the edge (Section 3).
— We pinpoint an in-network ensemble model well-suited for fast incident response and deployment within resource-constrained gateway devices on IoT edge (Section 4).
— We implement a federated-based model-sharing process for *in-network* ensemble model to preserve the privacy of local traffic information (Section 5).
— We highlight the importance of fast response to identified traffic anomalies and introduce an in-network solution expediting traffic handling to millisecond-scale on Raspberry Pi and microsecond-scale on a commodity switch (Intel Tofino) (Section 6).
— We present a framework prototype on Raspberry Pi and commodity switch hardware devices. Evaluation results on three public datasets show that our solution provides accurate and fast traffic analysis and response, with low overhead to traffic throughput and latency. Our framework outperforms compared baselines and SOTA, improving inference accuracy by 2%–17% and reducing communication overheads by 6%–60% (Section 7).

2 Background and Related Work

2.1 Background

Federated Learning (FL). FL enables the training of a global model across distributed local intelligent models while preserving privacy and reducing data transfer. Two types of FL have been studied, horizontal and vertical. Horizontal FL assumes that data presents the same feature groups, while vertical FL assumes that local data from different parties have distinct feature groups [57]. In this work, horizontal FL is studied by assuming that the same types of traffic features are collected in gateways. Considering ML models that can be federated, existing solutions have proposed frameworks to federate common-used models such as **Neural Networks (NN)** [33], ensemble tree models [29], **logistic regression (LR)** [14], and **support vector machine (SVM)** [26]. Among these models, NN-based models are widely used for their accurate learning performance on complex high-dimensional data. However, in traffic analysis services, ensemble-tree-based models have the advantage of using model structures less complex than deep neural networks, are computationally less intensive to train, and they outperform NN-based models in handling categorical features [15]. Therefore, ensemble-tree-based models can be a more suitable option for the analysis of raw packet bytes, which are tabular data with categorical features.

Programmable Data Plane. Traditional network devices have hard-coded data plane and control plane. While **Software-defined Networking (SDN)** has enabled flexibility in the control plane, recent years' development of programmable data plane has enabled flexibility within the data plane.

The common programmable data plane is based on the **Protocol Independent Switch Architecture (PISA)** as an abstract architecture. As shown in Figure 1, this architecture includes a packet header parser/deparser, and a control pipeline consisting of programmable logics and Reconfigurable Match-Action Tables (also known as M/A tables). P4 is a programming language used to define the fields in these blocks to instruct packet processing and forwarding [3]. Programmable data planes are supported on software and hardware target devices that pose different constraints. Software targets like *bmv2* are not constrained by processing architectures. Hardware targets that operate on ASICs have stringent restrictions on how resources are allocated within pipelines containing **Match Action Units (MAUs)** [20].

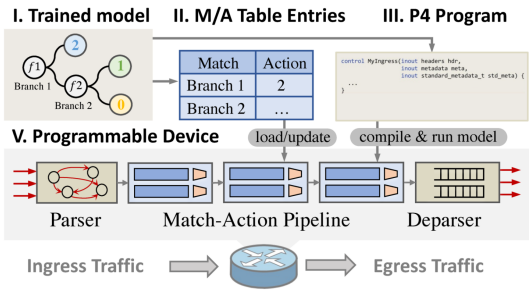


Fig. 1. Schematic PISA architecture for a programmable data plane in a programmable device (switch) [39]. The inference process of a tree-based ML model can be offloaded to the programmable data plane by mapping the trained model to P4 language and Match-Action (M/A) table rules.

In-network Machine Learning (ML) Inference. In-network computing is a concept where computational tasks are offloaded and performed directly within network devices (e.g., switches/routers) rather than sending all data to a centralized computing resource (e.g., server/cloud) for processing. In-network ML inference is an in-network computing service that specifically offloads ML inference to the network devices. With programmable logic in the programmable data plane, in-network computing is able to leverage the pipeline logic and computing resources in the network devices to perform fast and advanced network traffic processing.

Figure 1 demonstrates how in-network ML inference is achieved. Consider the high-level similarity between data flowing through the pipeline's tables in PISA architecture and traversing ML models, where both check the values of the current node and point to the next node. Consequently, ML inference process can be translated and mapped to PISA architecture when an ML model is well-trained based on datasets. Building upon this idea, researchers have explored commonly used ML models for in-network ML inference such as SVM [63], **Decision Tree, Random Forest (DT, RF)** [63], and NN [47].

2.2 Privacy and Fast Reaction Demands in IoT

As the number of IoT devices continues to grow, there is an increased need to provide low-latency services and improve service quality. IoT edge computing has been developed to process the data closer to the source. Rather than sending all data to remote servers/cloud, data can be processed at the edge of the IoT network. As illustrated in Figure 2, there are two types of "edge": on-premises edge and network edge. On-premises edge refers to an edge closer to end users that processes and analyzes data within users' local domain. Network edge indicates the edge of the operator's network that provisions resources within the network infrastructure, allowing for more advanced processing

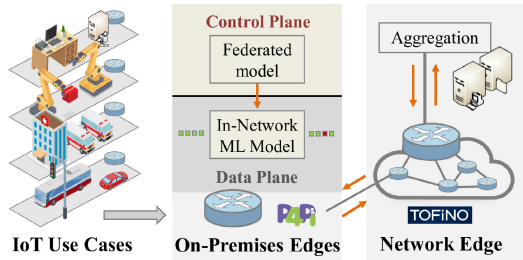


Fig. 2. Framework overview of FLIP4 on IoT edge.

and analytics. In this work, the network devices deployed on the on-premises edge indicate those devices with limited resources (e.g., Raspberry Pi) for directing data traffic within the local networks. The network devices deployed on the network edge indicate high-performance devices (e.g., commodity switch with Intel Tofino [16]) for handling substantial traffic.

Demands for fast reaction: 5G standards [10] have specified **Key Performance Indicators (KPI)** for time-critical services as listed in Table 1. These services require millisecond-scale latency, urging the demands for fast traffic analysis and quick response to issues and failures. With the fast and flexible processing performance of in-network ML inference in high-speed network devices, prior work has explored its applications on network edge deployment to enable fast traffic processing and response. Since offloading the ML-based analysis to the network edge reduces the processing latency, deploying in-network ML further accelerates the traffic analysis process on the data plane of network devices on IoT edge. However, in-network ML functions are deployed alongside network functions on resource-constrained IoT edge devices. This deployment needs to meet service KPIs while considering limited device resources.

Demands on privacy: When it comes to the IoT edge scenarios, privacy is another concern as many countries have released data privacy regulations [34–36]. Since ML-based analysis is a data-driven approach, data collection and sharing is inevitable for network operators or stakeholders to manage the applications and services. Though in-network ML provides intelligent analysis locally on the network edge, the dynamic nature of IoT traffic requires constant ML model maintenance to adapt to the variations and drift of incoming traffic. Such maintenance requires frequent traffic data sharing to provide a global view and learn the dynamics. It poses concerns about potential data breaches during the sharing or data interception for malicious profiling.

Motivating use cases: Three use cases demonstrate the aforementioned demands in realistic deployments.

(a) eHealth network. Wearable devices are applied to provide healthcare services to patients in remote locations, such as those recovering at home, with connections to hospitals. It is crucial to maintain the service performance across the network and to identify IoT devices and monitor their behavior for tasks like traffic profiling, identifying connectivity failures, or detecting anomalous traffic patterns [21]. However, handling sensitive information, such as network addresses and device details, raises privacy concerns among end-users.

(b) Vehicular network. Vehicular networks are dynamic networks connected with moving vehicles. The network is vulnerable to security threats if any vehicles are exploited to launch attacks on the network infrastructure. To detect potential security threats and prevent data breaches, traffic among vehicles can be analyzed at on-premises edge to trigger alerts and updates in real-time, contributing to vehicle safety and performance [44]. On-premises edge processing enables self-monitoring capabilities without relying on external servers causing high response latency. Privacy concerns could stem from activities such as unauthorized tracking and the exposure of sensitive data, such as vehicle locations.

(c) Campus network. Campus networks are highly dynamic, with multiple access points, and mobile users moving between them. The networks often pose security threats because of the number of mobile users. By identifying data-intensive applications, the network can allocate resources more efficiently and detect potential threats like malware, ensuring optimal performance [38]. Potential malicious interception of user data may become a privacy concern.

Table 1. On-premises Edge vs. Network Edge [41]

	On-premises Edge	Network Edge
Service Type	Data preprocessing	Data aggregation
Privacy	Sensitive user data breaches	Data interception and profiling
Latency	<5 ms	<10–40 ms
Data Rate	Mbps-Gbps	>Gbps
ML Capability	Limited	Medium

Table 2. Related Works of Traffic Analysis and Anomaly Detection in IoT Networks

Reference	Detection Model	Deployment Location	Distributed Intelligence	Detection	Mitigation	Implementation
[27]	GRU	Server	✓	✓	✗	Simulation
[45]	LR	IoT Devices	✓	✓	✗	Simulation
[6]	LR	Gateway control plane	✓	✓	✗	Simulation
[52]	NN	Gateway control plane	✓	✓	✗	Simulation
[24]	GRU	Switch control plane	✓	✓	✓	Simulation
[59]	DAE	IoT device	✓	✓	✗	Simulation, RPi
[40]	BNN	Switch data plane	✓	✓	✗	Simulation, SmartNIC
[58]	DT, RF	Gateway data plane	✗	✓	✓	Simulation, RPi
FLIP4	DT, RF, XGB	Gateway data plane	✓	✓	✓	Simulation, RPi, commodity switch

RF, Random Forest; LR, Logistic Regression; DAE, Deep Autoencoder; NN, Neural Network; MPL, Multilayer Perceptron; GRU, Gated Recurrent Units; SVM, Support Vector Machine; BNN, Binarized Neural Network; XGB, XGBoost.

Table 1 compares the detailed requirements between on-premises edge and network edge scenarios [1, 41]. To meet these requirements, deploying in-network ML inference on IoT edge devices for accurate attack detection and fast mitigation in different network locations has the following constraints:

(a) Computing resources: Network devices deployed on the on-premises edge are usually low-cost with limited computing resources. They are mainly deployed for traffic switching and routing. While gateways on the network edge are in charge of traffic processing toward the backbone or core network with higher throughput, the computing resources are less limited than on-premises edge gateways.
(b) Processing capability: Processing capability varies based on the amount of allocated bandwidth resources. For gateways deployed close to end users (on-premises edge), available bandwidth can range from Mbps to Gbps. While network-edge gateways are typically connected to high-speed cables or fibers with bandwidth resources from Gbps or more.
(c) Sensitive data processing: Depending on the application domain and geographic location, there might be regulatory restrictions on data processing and storage. Traffic information collected from gateways at on-premises edge may include sensitive information like the source and destination address of end users.
(d) Mobility: IoT end users may join or leave the network frequently, resulting in more challenges in accommodating to dynamic traffic from end devices at on-premises edge. Gateways on network edge have relatively static connections with mature link provisioning.

2.3 Related Work

Table 2 summarizes various related works in the field of FL-based attack defense from the aspects of the deployed models, deployment strategies, and their capabilities in detection and mitigation.

Distributed traffic analysis and handling. The heterogeneity and dynamic distributed deployment of IoT end devices like sensors and actuators have left vulnerabilities and attack vectors for anomalous activities. Traffic analysis has been studied to learn the traffic pattern and detect the potential risks in IoT networks [17, 45]. Most of the referenced works [6, 18, 59] include detection capabilities, demonstrating the efficiency of traffic analysis services identifying the anomalies in IoT networks. However, mitigation strategies to respond to detected anomalies are absent in most studies, which may result in delayed mitigation that could disrupt normal traffic and impact network infrastructure in low-latency communications. It highlights a gap in current research landscape.

Model complexity. To classify the traffic and identify the anomalous samples, some studies employ complex models like NN and GRU [24, 27, 52]. Despite complex models showing powerful

performance for various tasks, it might not be necessary or efficient in all IoT scenarios. This is especially the case for models deployed on IoT edge where network devices commonly have limited computing resources and may operate on low-power configurations. Considering that data is collected in tabular formats in traffic analysis tasks, simpler models can achieve comparable results with lower computational overhead. However, it is challenging to efficiently train and maintain the in-network ML in a distributed manner with low communication overhead.

Local model deployment strategy. Prior work deployed the local ML model on IoT devices [59] or gateway control planes [6, 18, 52], bringing extra communication overhead on traffic analysis where the collected network telemetry needs to be sent from the data plane to the control plane. This was limited by the constrained processing capability of traditional data plane architecture. By introducing the programmable architecture to the data plane of the network devices, ML model can be potentially deployed for inference in the data plane. Such potential allows fast traffic analysis for detection and fast mitigation once the inference decisions are made. Despite this potential, a few studies have explored the use of programmable architectures in data planes [40, 58]. However, prior studies like Reference [58] focused on hitless model updates in single-device deployment. They were not designed in the context of resource-constrained IoT scenarios or a distributed deployment.

2.4 Design Challenges

Though prior research has presented the advantages of applying FL to distributed traffic analysis in IoT networks, there remains a gap in fast countermeasures when analysis decisions are made. Deploying the analyzing algorithm on the data plane of IoT gateway may expedite the countermeasure process. It is challenging to apply the in-network model on IoT edge in a federated manner for knowledge sharing while adhering to the aforementioned constraints. This is due to several key factors:

(a) *Translating federated models to resource-constrained in-network deployment.* The model needs to be efficiently partitioned and translated to leverage the capabilities of the programmable data plane on resource-constrained network devices. This entails parsing the structure and parameters from the federated tree model and translating the parsed model information to P4-based in-network deployment.

(b) *Model aggregation with low communication overhead.* Model aggregation entails the information exchange to combine the local model parameters to create a global model for knowledge sharing. In the context of IoT deployment at on-premises edge devices, this process is ideally designed to minimize communication overhead to save the device resources.

(c) *Prompt response to inference decision.* When in-network ML is deployed for real-time analysis and decision-making, an agile and responsive workflow enables prompt actions based on inference decisions. Considering the potential trade-off between accuracy and privacy in FL, the workflow should be optimized to minimize the potential effect of false reactions, while allowing for timely responses to dynamic changes in traffic patterns.

Our design goal: A practical in-network FL framework for IoT edge in distributed scenarios that can achieve accurate traffic analysis with fast response, privacy-preserving traffic knowledge sharing, and lightweight deployment within resource-constrained gateway devices.

3 Proposed Framework

In this section, we present the proposed framework FLIP4. We first provide a description of the network scenarios and the threat model considered in this work. Following that, we introduce FLIP4's framework design, detailing its workflow and components.

3.1 Network Scenarios and Threat Model

Network scenarios. We consider a network setup with a server running on the cloud and N local clients running in edge domains. Note that these clients are not end-users in this work; instead, they are network devices that support a group of local end-users. As in the scenarios discussed earlier, these local clients act as gateways, which can be strategically located either at the on-premises edge or at the network edge (as illustrated in Figure 2). Depending on specific demands and characteristics of the respective edge locations, packet-based and flow-based features are extracted by local clients deployed as gateways at the on-premises edge and network edge. These extracted features are then utilized by these gateways to conduct ML-based traffic analysis, focusing on traffic classification tasks.

Threat model. We assume the attackers have already gained access to the network with IoT devices using obtained credentials. This compromised network is based on the frequent scenarios where the devices are misconfigured or low-security configuration to lower the power consumption [48]. This work focuses on attacks exploiting network protocols, specifically IoT edge deployment scenarios involving gateway connections. (1) We concentrate on analyzing issues at the protocol level, which can arise during protocol communications and require investigation through traffic analysis. Even in cases where traffic is encrypted, L2–L4 protocol headers remain in plaintext, retaining their visibility for analysis. (2) We assume the network devices running the framework are not compromised. (3) We consider the threat that attackers could potentially launch differential attacks.

3.2 Framework Overview

Design choices. FL involves both local devices as the clients and servers/cloud for model aggregation. To enable accurate traffic analysis with fast reactions within distributed IoT gateways and to meet the demands in IoT network discussed in Section 2.2, we consider the following design choices: (a) Local IoT gateways: Local gateways are integrated with a programmable data plane enabling in-network traffic analysis and fast reaction function. These gateways with programmable data planes can be programmed with P4 language for flexible traffic processing and efficient traffic handling. Depending on the location of the gateways, they can run on RPi deployed at the on-premises edge or high-performance switch deployed at the network edge. On each gateway, the control plane runs on top of the data plane instructing the data plane functions and coordinating the FL process on the server. (b) Server: The server provides FL-related services, such as model parameter's collection and aggregation for privacy-preserving knowledge sharing across local gateways.

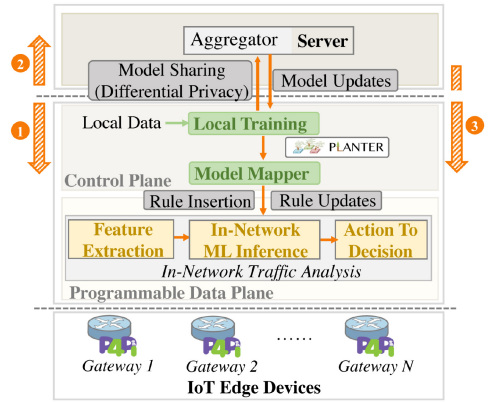


Fig. 3. Overview of the proposed framework.

Proposed workflow. Detailed workflow is depicted in Figure 3. The framework is composed of a central server and multiple IoT gateways. Each gateway runs the programmable data plane and control plane. The workflow consists of three main steps: in-network inference, FL-based model training, and runtime updates.

— **In-network inference:** One of the key challenges in enabling fast incident response is the delay caused by the separation of traffic forwarding (data plane) and ML model training and inference for analysis (control plane/server). This separation leads to delays in analyzing and

acting upon detected incidents from incoming traffic. The proposed framework addresses this challenge by offloading the ML-based inference from the control plane/server directly to the data plane, in what we call "in-network inference" (as Figure 3 step ❶). It is done within the network data plane so that the incoming traffic is parsed and analyzed at the forwarding path based on the in-network ensemble tree model deployed at the data plane. In detail, 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 ML-based analysis and labeling. If a packet is labeled as benign, then it will be forwarded; otherwise, it will be dropped or metered.

— **FL-based model training:** When the in-network inference is deployed in each local gateway for fast attack detection and mitigation, it raises the challenge of sharing local knowledge for optimal global analysis without incurring communication overhead or risking data leakage. To address this, we introduce FL to share local model parameters among multiple gateways, rather than the source data itself, to train a global model at the server (as Figure 3 step ❷). When the server receives the model information from all gateways, the aggregator starts to aggregate the received models to generate a global model. Differential Privacy is introduced in this process to improve privacy.

— **Runtime updates:** Another challenge arises in updating the local ML models with aggregated global model without disrupting the functioning of local IoT gateways. Our solution involves runtime updates to the local models for optimal accuracy, after the global model is disseminated from the server back to each gateway (as Figure 3 step ❸). At each gateway, a mapper coordinates between the control plane and data plane, updating the M/A table rules with the new model. These rules are then seamlessly integrated into the data plane, ensuring uninterrupted traffic flow while updating the system with the latest aggregated knowledge.

Framework components. The framework is structured around two core components: the mapper and the aggregator. Each IoT gateway hosts a mapper responsible for local model mapping. This process involves converting the model's parameters into M/A table rules for in-network inference, which are dynamically inserted into the data plane. The mapper also shares these trained model parameters with the server in a privacy-preserving manner with Differential Privacy and authentication measures. On the server side, the aggregator will generate the global model. Once it receives the updated model parameters from all the gateways, it begins the process of averaging these parameters to construct a global model. This global model's parameters are then distributed back to each local IoT gateway. The gateways use these parameters to update their local models, ensuring that they are synchronized with the latest global insights and trends. This cycle of local inference, model sharing, and global model updating forms the framework, enabling fast defense services and privacy-preserving knowledge sharing.

The proposed framework brings the following **benefits**: (1) in-network models provide enhanced accuracy and fast defense to identify complex attack patterns that traditional rule-based solutions might miss with predefined rules on the data plane, (2) FL-based training method ensures knowledge sharing and reduces communication overhead, and (3) the periodic model aggregation and runtime model update enable automatic local model reconfiguration and dynamic adaptation in response to changing threats in the network.

4 Local In-network Traffic Analysis

In this section, we demonstrate how in-network inference is designed for efficient local traffic analysis and fast incident response. We first introduce how the features are extracted from the arriving traffic and how these features are input to the local ensemble model for in-network traffic

analysis. We explain the benefits of selecting ensemble tree models and present the challenge of implementing the typical ensemble model (XGBoost) on the resource-constrained data plane, as well as how to overcome this challenge to realize the model on the programmable data plane.

4.1 In-band Traffic Feature Extraction

To achieve in-network traffic analysis locally in each distributed gateway, features are 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. In this work, stateless and stateful features are extracted based on the feature importance ranking results in different use cases. For instance, flow-based attack detection includes stateful flow-based features and IoT fingerprinting includes stateless features. Considering the network traffic is parsed in tabular format, *Permutation Importance* is used to compute the feature importance by the degree to which the model performance score decreases after the feature is randomly shuffled [4].

Having the features selected based on the ranking results, the selected features are extracted in an in-band manner on the data plane. The stateless features are extracted by directly parsing the packet headers from the arriving traffic. Stateful features (e.g., TCP flags, inter-arrival time) are collected and counted with counters and registers and concatenated in the *bitstring* data structure based on five-tuple flow information [5].

4.2 Selection of In-network ML Models

To attain the goal of quickly responding to traffic by using an ML-based traffic analysis method, ML models can be deployed in an in-network manner within the local network devices (gateways). This allows the model to perform inference concurrently with the traffic switching process on the data plane, enabling direct ML traffic control within the network device. Many ML algorithms, which are suitable for distributed training, can be used for FL. However, not all these algorithms are suitable for in-network deployment, especially in terms of accuracy on the resource-constrained programmable data plane [62]. Different from the typical ML, the trained model can be directly loaded to the processors with a wide range of operations supported, the in-network ML model needs to be translated and mapped to the M/A form that can be processed by programmable pipelines. Thereby, the model performance may be limited by the computing resources on network devices and may lead to compromised accuracy.

Prior work has presented that some algorithms such as DT, RF, **XGBoost (XGB)**, ***k*-means (KM)**, ***k*-Nearest Neighbors (KNN)**, **Naive Bayes (NB)**, and SVM, have the potential to be implemented [64]. However, there is a lack of discussion and a challenge in selecting a proper model for accurate and low-overhead in-network deployment of traffic analysis service on IoT edge. When deployed on a hardware target like a commodity programmable switch on the network edge (more user connections and higher process requirements), resource bottleneck may come from the maximum capacity of the hardware target such as pipeline stages, memory, processor, and so on. Such constraints affect the performance of in-network ML inference under the resource requirements in different use cases [5, 19, 61]. To select a proper model, we examine the inference performance and resource efficiency (in terms of pipeline stages and M/A table entries) of these models deployed in data plane for attack detection service.

Table 3 shows the accuracy evaluation results of common-used ML models implemented in an in-network manner. Public dataset CICIDS 2017 [46] is used as an example to compare the attack detection performance in scenarios like vehicular network [56]. The constrained resources on hardware targets can reflect the general performance of P4 program in the data plane [64]. The

Table 3. Model Comparison: In-network Implementation of ML Models on CICIDS 2017 Dataset

Model	DT	RF	XGB	KM	KNN	NB	SVM	NN
ACC	99.87	99.81	**99.91**	58.40	64.43	99.48	86.54	92.00
F1	99.86	99.80	**99.91**	56.80	50.61	99.46	86.44	92.00
Stages	**2**	3	3	7	**2**	8	9	»12
Entries	**116**	487	1k	132k	14k	132k	132k	0

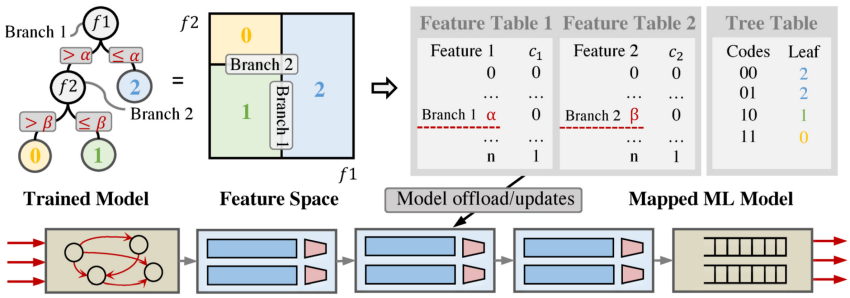


Fig. 4. Mapping process of tree-based model to a programmable data plane. A single decision tree is depicted as an example. XGBoost is an ensemble of trees.

table illustrates an observation: DT and its ensemble models, especially XGBoost, exhibit the best ML performance.

4.3 In-network XGBoost Implementation

Feature extraction in the data plane of the local IoT gateway allows the concurrent execution of ML model inference with network functionality, enabling a swift response to incidents. By enabling the ML model inference process within local switches (specifically, to the switch's data plane, where traffic is routed), the time taken for traffic data collection and analysis can be significantly reduced by directly identifying the anomalies on the forwarding path within the programmable switch. Thereby, incident responses can be executed promptly within the programmable switch's pipeline once the anomalous incident is identified, as depicted in Figure 1. This immediate analysis and reaction in an in-network manner is critical for mitigating detected malicious activities, preventing them from further affecting other parts of the network. This subsection introduces implementation details of XGBoost model in an in-network manner.

XGBoost is ensembled based on basic DT models. Each DT model acts as a weak learner and is trained sequentially to correct the errors made by previous learners. When it comes to the data plane, even a simple DT needs to compromise depth due to the excessive amount of stages. To better realize tree models, an implicit mapping is necessary. Figure 1 depicts a simple example of how a tree model can be translated to a P4 program and a set of M/A tables, as well as shows the deployment process toward the programmable data plane. Figure 4 presents the details of the model mapping process in P4 program. Since the trained tree model can be regarded as splits of feature space, each branch can be treated as the boundary between split areas. Each split area is labeled by the value in the leaf node. When learning an input consisting of n features, a single tree model is mapped to the data plane with n feature tables and a decision table. As a result, the inference process starts with the feature table, where each input feature is associated with specific codes, and each code represents a threshold range corresponding to that feature. Subsequent to

the feature table, the decision table retains the associations between the codes derived from all features and the values linked to the leaf nodes.

In XGBoost, the weight is represented by the value stored in the leaf of each tree model. These weights are usually non-integer values, which are not supported by the P4 program. To tackle this challenge, we choose to encode each weight into the weight code. We apply a forest table to map all the weight codes from each tree model to the output class. By doing the encoding process, feature tables and tree tables can be incorporated into the pipeline in parallel. The resource consumption in terms of pipeline stages of P4 program can be thus effectively optimized within the pipeline.

5 FL-based Collaborative Model Training

In this section, we explain the details of the FL-based training for in-network local models and describe the workflow for federating local models into a global model. First, we identify the challenges of ensemble FL compared with the federated NN. Second, we introduce the workflow in detail to illustrate how the in-network local models can be federated to a global model to share the traffic information in a privacy-preserving manner.

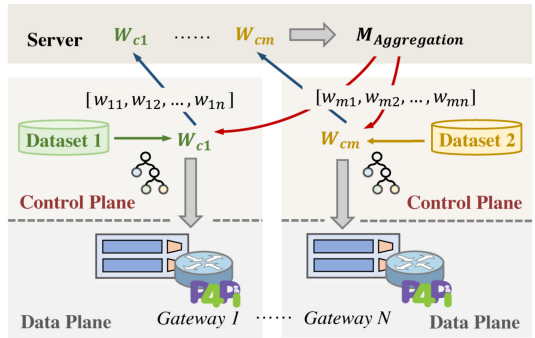


Fig. 5. Sample workflow of FL function for model sharing and parameter updates.

5.1 Federating Ensemble Models

Different from NN-based model aggregating the weights and biases of each layer, federated tree models update the weights of leaf nodes in each ensemble tree. Such difference brings distinct challenges in model aggregation and translation to local in-network tree models.

The challenges of aggregating local tree-based models to a global model are: (a) Tree model may grow asymmetrically during the training process, unlike NN with a symmetric layer-by-layer model structure. Thereby, sharing the model parameters needs to bind the tree node information together, which includes the feature splits and the values in each tree node; (b) Aggregating a global model in the server requires rebuilding the tree to update the global information instead of a simple value assignment.

We address these two challenges by incorporating the federated workflow. In this workflow, the local models are shared with both parameters and structure information to the server. The server triggers the tree rebuilding process upon receiving the information. The overall federated workflow can be divided into the following steps:

Local training: Each client (local gateway) initializes the tree model locally by training its local data. The training process involves recursively building a local tree model, similar to standard tree-based model training.

Model aggregation: The central server first initializes an empty tree structure as global tree model. It then collects local models from all gateways participating in the FL process. It aggregates these models to obtain global model.

Model distribution: The updated global model is then distributed back to clients. This step involves the updates of in-network model parameters at runtime, enabling local in-network model updates without disrupting normal traffic.

Periodically, the process continues with the next round of local training. The whole process involves multiple rounds of training and aggregation to improve the global model's performance gradually as described in Algorithm 1.

ALGORITHM 1: Federated tree to in-network tree

```
On server:
for r in Rounds do
    for e in Epoches do
        W = Model aggregation
    end for
    for n in local gateways N do
        UpdateLocalModel
    end for
end for
Broadcast globalModel to all gateways
Output globalModel
On local gateway:
procedure UpdateLocalModel
    for r in Rounds do
        Parse model parameters W_avg in globalModel
        Generate new M/A rules
        Insert rules to data plane of local gateway at runtime
    end for
end procedure
```

5.2 Efficient Model Aggregation

Aggregation workflow. Figure 5 depicts a sample workflow of the FL function proposed for in-network deployment. The process begins with the server initializing a model structure and sharing it with N local devices. With this information, each local device initializes the model and proceeds to train its own model using its respective local data. After the training process, parameters of the trained local model $[w_{11}, \ldots, w_{1n}]$ are packaged and transmitted from each device to the server. In order to enhance the global model, the server aggregates the local models from each local device through a weighted average mechanism. The aggregated model $M_{Aggregation}$ is then transmitted back to the local devices. Once each local device receives this aggregated model, it will utilize these new parameters to update the model. This update involves the adjustment of decision boundaries within the tree structure and the generation of new M/A table entries for the data plane. Compared with methods that hard-code the model parameters in P4 program, mapping the model to table entries provides the flexibility to change and update the model during runtime, building upon the architectural advantages of the programmable data plane. Details are discussed in Section 6.

Privacy-preserving aggregation with Differential Privacy. We use the common aggregation method named **Federated Averaging (FedAvg)** [30] in this work. It aggregates model updates from multiple local devices while preserving data privacy. As shown in Algorithm 2, the formula for FedAvg is represented as follows: $W_{avg} = \frac{1}{N}\sum_{i=1}^{N} w_i$, where we have N local gateways in the federated setup and each model has its own model weights W_i, where $i = 1, 2, \ldots, N$. Global server benefits from the knowledge of all devices' local models without directly accessing local data, ensuring data privacy during traffic analysis process. Aggregating ensemble boosting tree model with a set of data samples $D = \{(x_i, y_i)\}_{i=1}^{n}$ is based on the objective function of tree f denoted as: $\mathcal{L}(f) \approx \sum_{i=1}^{N} (\ell(y_i, \hat{y}) + g_i f(x_i)) + \Omega$, where l is the leaf node, $\hat{y}$ is the prediction of x, g_i is the gradient of the loss function, and $\Omega = \gamma L + \frac{\lambda}{2}\|\boldsymbol{w}\|_2^2$. Model parameters updates involve updating the gradient weight $w_l = -\frac{\sum_{i \in I_l} g_i}{|I_l| + \lambda}$ [29]. The weights of leaf nodes contribute to the ensemble vote and affect predictions. Model updates therefore aim to update such weights in leaf nodes.

ALGORITHM 2: Federated tree aggregation with Differential Privacy

Input: Model parameters from local gateways $w_1, w_2, \ldots, w_n$
Output: Federated tree model M
procedure FederatedAggregation(*parameters*, *privacyBudget*)
    $globalModel \leftarrow$ InitializeModel()
    $modelUpdates \leftarrow []$
    for each parameter set w_i in gateways **do**
        for $tree_j$ in w_i **do**
            Recompute $tree_j$ tree splits in each branch
            Use new parameters to rebuild $tree_j$ in w_i
            Recalculate weights for $tree_j$: $W_{avg} = \frac{1}{N}\sum_{j=1}^{N} w_j$
        end for
        $noisyUpdate \leftarrow ApplyDP(localModel, privacyBudget)$
        $modelUpdates.append(noisyUpdate)$
    end for
    $globalModel \leftarrow Aggregate(modelUpdates)$
    return *globalModel*
end procedure
procedure ApplyDP(*model*, *privacyBudget*)
    Compute privacy budget ϵ.
    Add noise $\eta \sim \mathrm{DP}(\epsilon)$ to the update: $\tilde{M}_i = M_i + \eta$
end procedure

Differential Privacy (DP). As a transmission of model's parameters may be intercepted by differential attacks and leak the model information, DP is introduced in this work to the model-sharing process by adding randomness/noise to the model outputs. DP can quantify and manage the degree of the privacy risks of the model by statistically characterizing the impact of a single data element on the model. It is based on the theorem that the added randomness $\mathcal{K}$ satisfies ϵ−DP, defined as:

$$\Pr\left[\mathcal{K}\left(D_1\right) \in S\right] \leq \exp(\epsilon) \times \Pr\left[\mathcal{K}\left(D_2\right) \in S\right], \tag{1}$$

where D_1 and D_2 are any adjacent datasets and and all outcomes $S \subseteq \mathrm{Range}(\mathcal{K})$ [9]. Under this definition, the impact of a single data element on the model is controlled within a certain range by making it less distinguishable for differential attacks at the model level. The concept *privacy budget* is used to evaluate the performance of the differential privacy mechanism and it is expected to be as small as possible for better privacy. 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 [32] is used in this work by adding Rényi divergence to provide more fine-grained control of privacy while keeping the succinctness.

Low-overhead communication with authentication. To exchange the model information with low overhead, we select a group of candidates to share the parameters of local models, instead of obtaining information from all clients. The candidate selection is done randomly in each iteration to reduce the communication overhead. To keep the communication lightweight, we configure the web socket as the communication technique to transmit the model parameters. Compared with API-based communication, the socket-based solution runs at a low level with lower communication overhead, despite the shared information being less customized. Besides, to prevent potential model poisoning from the attackers [43], authentication is configured with socket-based communication to only allow model sharing from the authenticated users. Based on usernames and passwords hashed in a private *JSON*, the central server compares the credentials

from the local devices to determine whether the local devices are authenticated to share model information.

6 In-network Traffic Analysis and Fast Reaction

After the model distribution, local in-network models are configured at runtime to support accurate and fast traffic analysis. Deploying such an in-network approach offers advantages in rapidly responding to inference decisions and meeting requirements for time-critical service on IoT edge.

Runtime configuration. Typical model updates involve the reloading of the model parameters. In the context of in-network ML inference, the update can be problematic as the model parameters have been inserted in the M/A tables inside the data plane processing pipelines. Updating the model interface may result in disruption of the forwarding functionality. Thereby, P4Runtime interface is configured between control plane and data plane to enable seamless updates of the model parameters without affecting the normal traffic. When local devices receive the model parameters from the global model, the runtime updates take place by first generating the new set of M/A table rules to map the parameters in the tree structure. By writing the newly generated rules, representing updated model parameters, to tables in the data plane via P4Runtime interface, the in-network ML model is updated at runtime without interrupting the forwarding functions. Incoming traffic can thereby obtain the latest ML-based inference results as it traverses the pipeline.

Fast reaction. When inference decisions are made on the data plane of the IoT gateway, an action is conducted to instruct traffic handling. Such an action taking place in the data plane, right after the inference decisions, can enable fast incident response. In this work, we propose three actions: forward, meter, and drop. It enables the following use cases. In the anomaly detection use case, filtering the detected malicious traffic by dropping is an efficient way to lower its potential impact on other parts of the network. Yet, when FL is introduced to preserve learning privacy, there is a trade-off between accuracy and privacy due to the indirect data-sharing process. Thus, simply dropping the detected malicious traffic may affect the network service if the inference gives false alerts. Thus, traffic metering allows a moderate operation to throttle the suspicious traffic, which limits the impact of the malicious traffic and secures the reaction from false alerts. This also applies to other services like flow classification or QoS management where metering is needed to implement flow scheduling to improve resource utilization. The configuration of the three actions is chosen based on the security level of the service. If the security level is high, then a drop action is configured to ensure the detected threats are mitigated. A meter action is used when the traffic's service level needs to be guaranteed, avoiding the impact of false alerts.

7 Evaluation

7.1 Experiment Settings

Experimental setup We implement our prototype framework in P4 language using *bmv2* with v1model architecture. Two types of deployment were prototyped. For on-premises edge deployment, the proposed solution is run on Raspberry Pi using P4Pi-v.0.0.3 [49] to play as the IoT gateway. As for the IoT network-edge deployment, programmable switch APS-Networks BF6064X with Intel Tofino 1 chipset and Barefoot's SDE 9.6.0 is used to test the performance in commodity high-throughput hardware. While P4 code provides the main data plane functions, Python code provides controller and server functionality, extending the design in FL algorithm [29] and *Planter* [64] for model training and inference. To evaluate the scalability of the proposed framework and be limited by the hardware setup, Mininet is used to emulate a network with a number of devices and gateways. To compare the learning performance, the performance of the proposed framework is compared with other ML models as baselines, where baseline results are taken from offline model learning using NN in *PyTorch* and tree-based ML models in *sklearn*.

Network setup: FLIP4 is deployed for a distributed network scenario on IoT edge. We consider both on-premises edge and network-edge deployment scenarios. For an on-premises edge network scenario, a SOHO (Small Office/Home Office) network is configured where a limited number of gateways are connected. For a network-edge scenario, a mobile edge network with edge computing capability is considered where traffic is processed at Gigabyte- to Terabyte-throughput. To simulate the multiple-access gateways, a network topology is built in Mininet to the network edge with multiple switches acting as gateways. These gateways are connected to the remote server via switches. The number of devices in the topology varies between the experiments.

Datasets: Three public datasets with packet-level features or flow-level features are used to evaluate the performance of the proposed solution in different services. All three datasets include both CSV and PCAP files. The CSV files are used for training, while the PCAP files are replayed in the performance evaluation of FLIP4 on the hardware prototype.

- IoT Sentinel [31]: A device fingerprinting dataset for identification including packet features for IoT identification service. Records from laptops and cameras are used where the samples are balanced with a ratio of 49%.
- CICIDS 2017 [46]: An intrusion detection dataset including flow-based features for attack detection service. Records on Wednesday including several types of DDoS attacks are used where the benign/malicious ratio is imbalanced at a ratio of around 90% of malicious traffic.
- CIC-AndMal2017 [22]: A malware dataset recorded by flow-level traffic features for malware detection services. Records on Ransomware samples are used where the benign/malicious ratio is imbalanced at a ratio of around 14% malicious behavior.

7.2 Evaluation Metrics

Learning accuracy: Several metrics are available to evaluate the learning performance of ML models. In this article, we use **Accuracy (ACC)** and **Receiver Operating Characteristic (ROC)** Curve as the metrics to evaluate learning accuracy performance, while calculating the **area under the ROC Curve (AUC)**. They are defined as below, where True-positive Rate $TPR = \frac{TP}{TP+FN}$ and False-positive Rate $FPR = \frac{FP}{TN+FP}$. F1 score is also used $F_1 = 2 * \frac{Precision*Recall}{Precision+Recall}$, where $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$:

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

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

Latency: To evaluate whether in-network ML inference brings extra latency to gateways, latency is measured as packet processing latency within IoT gateway devices. It indicates the duration when packets traverse the processing pipeline and is computed by $T_L = T_{Egress} - T_{Ingress}$.

Throughput: To evaluate whether the in-network ML inference brings any overhead to the gateway, network throughput is measured by *iPerf2* on P4Pi and *ucli* on Tofino.

CPU resources: The consumption of CPU resources is tracked to measure the impact of the proposed solution on the gateway's workload. To do this, CPU utilization is logged using the system information tool */proc/stat* on the P4Pi platform, providing how CPU cycles are used across each core. The tool *vcgencmd* is employed to collect data on the core temperature.

Table 4. Detection Accuracy on CICIDS 2017 Dataset

Model	Gateway1	Gateway2	Gateway3	Global	Offline
KM [12]	0.826	0.655	0.577	—	0.651
KNN [64]	0.875	0.386	0.432	—	0.432
NB [63]	0.899	0.693	0.967	—	0.645
NN [64]	0.985	0.936	0.968	—	0.930
F-DT [28]	0.899	0.500	0.500	0.788	0.977
F-BNN [40]	0.917	0.681	0.971	0.900	0.920
FLIP4-XGB	0.946	0.782	0.964	**0.919**	**0.988**

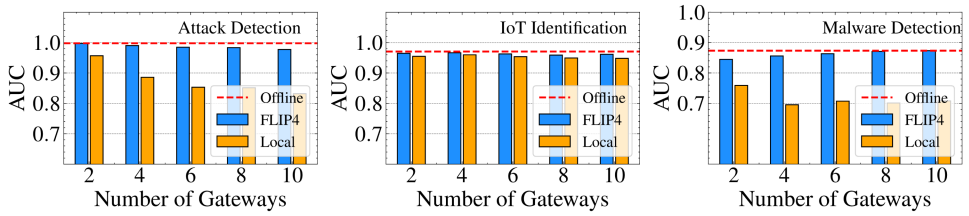
Table 5. Detection Accuracy on IoT Sentinel Dataset

	Gateway1	Gateway2	Gateway3	Global	Offline
KM [12]	0.440	0.448	0.473	—	0.545
KNN [64]	0.563	0.500	0.510	—	0.543
NB [64]	0.787	0.797	0.780	—	0.792
NN [64]	0.957	0.970	0.940	—	0.958
F-DT [28]	0.942	0.944	0.944	0.947	0.962
F-BNN [40]	0.937	0.954	0.936	0.936	**0.968**
FLIP4-XGB	0.958	0.953	0.959	**0.962**	0.966

7.3 Inference Performance

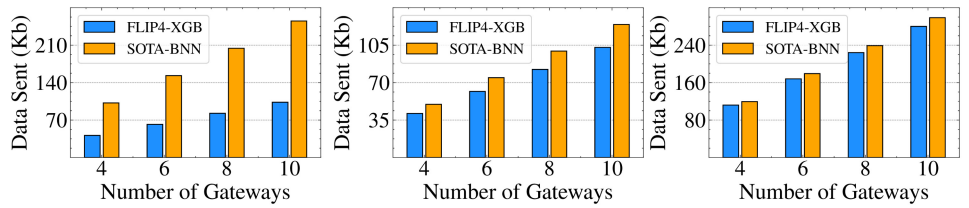
Inference accuracy. Tables 4 and 5 summarize the inference accuracy of FLIP4 using *CICIDS 2017* [46] and *IoT Sentinel* [31] dataset, respectively. They list the comparison of inference performance between XGB model of FLIP4 and SOTA work BNN [40] implemented in an in-network manner. Other ML models—KM, KNN, NB, and NN [12, 63, 64]—are also listed showing the baseline accuracy on source data at the server without FL functions. The tables summarize federated performance results of local inference performance at each gateway and the global model performance in server. Results show that FLIP4 can reach similar accuracy performance as the baseline NN-based model and SOTA work with the federated in-network BNN model in both datasets. It outperforms other classical ML models like NB. Detailed observations are: *(a) Local performance with in-network deployment.* Compared with the uneven accuracy performance of other classical ML models on local gateways, in-network deployment of XGBoost in FLIP4 presents relatively balanced high accuracy. This is especially the case in IoT Sentinel dataset, which is because of XGB's advance in ensemble-based decision and averaged results from FL. *(b) Global performance with FL.* The aggregated global model of XGBoost in FLIP4 shows higher accuracy performance in both datasets than prior SOTA FL work [28, 40]. In detail, XGBoost in FLIP4 performs 2%–3% higher accuracy than DT-based [28] and BNN-based [40] SOTA in IoT sentinel dataset. In CICIDS 2017 dataset, XGB proposed in this work has a similar level of advantage to BNN-based design [40] while showing 17% higher accuracy than DT-based FL [28]. *(c) Global performance versus offline performance.* Owing to privacy measures, FLIP4's global accuracy is slightly below that of offline results obtained through direct training on source data. The accuracy degradation varies between datasets, from 0.4% in IoT Sentinel dataset to 6% in CICIDS 2017. Nevertheless, it surpasses the global accuracy of SOTA models like F-DT [28] and F-BNN [40].

Scalability. Figure 6 extends the experimental setup in Table 4 and evaluates how FLIP4 scales to a different number of gateways on IoT edge. Evaluation is conducted on three datasets, and



(a) AUC vs. Gateway Num. (b) AUC vs. Gateway Num. (c) AUC vs. Gateway Num.

Fig. 6. FLIP4 performance on learning accuracy as AUC score vs. number of gateways. Offline—offline model trained by source data, FLIP4—global model, Local—local model. The datasets used are (a) Attack detection in *CICIDS 2017* [46], (b) IoT Identification in *IoT Sentinel* [31], and (c) Malware detection in *CIC-AndMal2017* [22].



(a) Data Sent vs. Gateway Num. (b) Data Sent vs. Gateway Num. (c) Data Sent vs. Gateway Num.

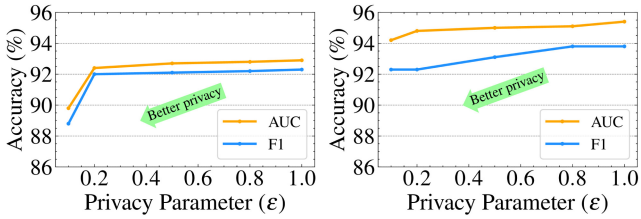
Fig. 7. FLIP4 performance on data sent vs. number of gateways. Offline—offline model trained by source data, FLIP4—global model, Local—local model. The datasets used are (a) Attack detection in *CICIDS 2017* [46], (b) IoT Identification in *IoT Sentinel* [31], and (c) Malware detection in *CIC-AndMal2017* [22].

performance is compared between the average local inference accuracy versus the global accuracy aggregated by FLIP4. The red dashed line marks a baseline accuracy of XGB in an offline manner on a server. Results present FLIP4 scales well to multiple gateways. As more gateways are connected as new domains, a slight trend of accuracy degradation is shown in CICIDS 2017 dataset. Such degradation is caused by data distribution of local datasets.

Communication overhead. With the same experimental setup as Figure 6, Figure 7 illustrate how the volume of data may vary as more gateways promote their local model to the central aggregator. In this group of figures, as the number of gateways increases, the number of models being transmitted to the aggregator also increases, resulting in an increasing amount of data to be shared for global integration. Results show that tree-based model deployed in FLIP4 can successfully lower the communication overhead. In CIC-AndMal2017 dataset, FLIP4 reduces 6.2% of data sharing volume than BNN-based solution [40]. This reduction is more significant in CICIDS 2017 dataset where the data volume is reduced by 60%.

Model training time. With the experimental results in Figure 6, the model training time for local gateways is ~0.01 s, and the global integration and update for the server is ~0.71 s. To enable and trigger in-network ML inference, the controller requires ~0.11 s to insert table entries to the data plane. During this experiment, a 15 s time window is established for global aggregation with five local gateways connected. The window size configuration may differ based on the number of gateways involved. This is because the aggregation process is activated once all gateways complete the model-sharing process to ensure a reliable outcome.

Accuracy versus privacy trade-off. Introducing Differential Privacy into the model-sharing process enhances privacy by adding noise, but this may impact the accuracy of inference. Figure 8



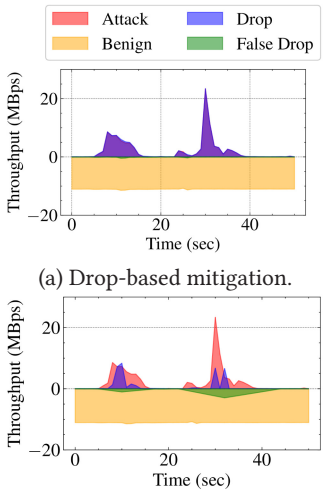
(a) CICIDS 2017 dataset. (b) IoT Sentinel dataset.

Fig. 8. Privacy-preserving performance by varying ϵ.

presents empirical results of how increasing privacy affects inference accuracy. The privacy parameter ϵ (as defined in Equation 1), quantifies the privacy budget. A lower ϵ value corresponds to better privacy. By decreasing ϵ to give better privacy, accuracy trends decline. Specifically, there is a 2.7% accuracy decrement in the CICIDS 2017 dataset and a 1.4% accuracy decrement in the IoT Sentinel dataset when ϵ varies from 1 to 0.1. The curves hit a pivotal point at $\epsilon = 0.2$, where a noticeable drop in accuracy occurs as privacy improves. This indicates that setting ϵ higher than 0.2 can yield a better trade-off between accuracy and privacy, leading to less privacy loss.

7.4 In-network Reaction Performance

With the software experimental setup in Figure 6(a), Figure 9 presents the evaluation of reaction performance when FLIP4 is deployed for in-network traffic processing. Figure 9(a) demonstrates that FLIP4 enables immediate mitigation action within milliseconds as the blue area (traffic being dropped) overlaps with the red area (malicious traffic). Prior in-network solution [23] adopts direct dropping on suspicious traffic to kill potential anomalies as soon as possible. In this work, considering the aforementioned performance degradation in accuracy, which is the price of privacy, another metering-based measure is deployed to limit the rate of detected malicious flows. This ensures that false alert traffic will not disrupt the normal service traffic [37, 54]. Results indicate that metering-based methods reduce throughput impact from volumetric attacks compared to the drop-based method, which immediately discards all identified anomalies.



(a) Drop-based mitigation.

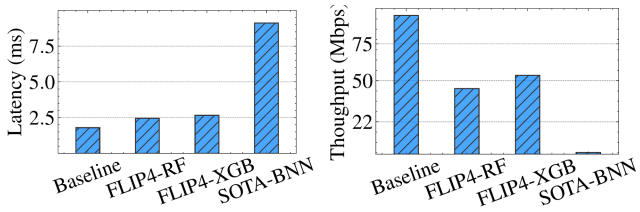
(b) Metering-based mitigation.

Fig. 9. In-network traffic mitigation of FLIP4.

7.5 System Performance

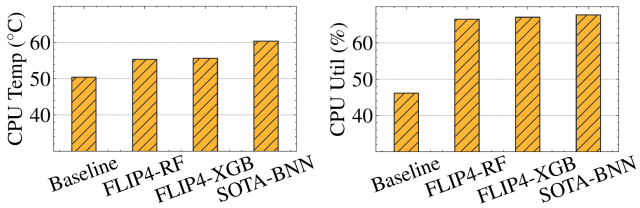
Inference speedup. We measure the processing latency in an IoT gateway to assess how this in-network ML approach affects normal traffic on data plane. Figure 10(a) presents latency tests of FLIP4 and SOTA [40], with the same experimental setup in *bmv2*. Baseline represents a basic gateway status without any analysis approach deployed. Results show that FLIP4 distinguishes itself through minimal latency overhead. While FLIP4 achieves packet processing latency of 2 ms, SOTA approach introduces ×5 latency to 10 ms.

Lightweight deployment. Figure 10(b) presents throughput performance of FLIP4 deployed on P4Pi [49]. Baseline shows a scenario when only packet-switching functions are enabled in gateway. FLIP4 and SOTA-BNN [40] present the scenario when in-network ML inference deployment



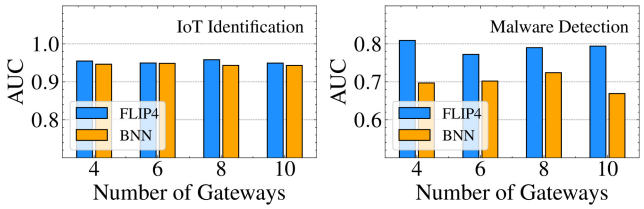
(a) Latency comparison. (b) Throughput comparison.

Fig. 10. Latency and throughput comparison between FLIP4 and SOTA work BNN [40] in IoT Sentinel dataset.



(a) CPU temperature. (b) CPU utilization.

Fig. 11. Impact on CPU resource consumption.



(a) IoT Sentinel dataset. (b) CIC-AndMal2017 dataset.

Fig. 12. Accuracy performance comparison between FLIP4 and BNN [40].

proposed in this work and SOTA. While SOTA presents severe throughput degradation by more than 90%, presenting only 2 Mbps throughput, the deployment of FLIP4 causes less throughput degradation with 54 Mbps when in-network XGB is deployed in data plane for traffic analysis. That shows tree-based model used by FLIP4 causes less burden to gateway than BNN-based SOTA. This is due to the complex structure and operation of BNN, which could potentially overload devices. When it comes to impact on CPU resources on RPi-based deployment as in Figure 11, despite with similar level of CPU utilization, FLIP4 causes 50% less burden than the SOTA on CPU temperatures.

When it comes to deployment on Intel Tofino commodity switch, the tree-based solution in FLIP4 also shows an advantage with line-rate speed at 6.4 Tbps throughput with sub-microsecond latency, while BNN-based solution reaches the maximum resource limitations and fails to achieve line-rate throughput. In detail, running XGB in FLIP4 consumes 3 pipeline stages, while running SOTA-BNN cannot be supported in maximum 12 stages due to model complexity. Commodity programmable device has limited resources and is strict on processing operations and resource usage.

7.6 Comparison with State-of-the-Art

Figure 12 presents the accuracy performance compared with SOTA solutions, it shows that FLIP4 shows similar scalability as SOTA-BNN [40] in CICIDS 2017 dataset, but performs better to scale in CIC-AndMal2017 dataset. When it comes to the system performance, the comparison is presented in Figures 10 and 11 where FLIP4 causes less effect on the forwarding functions and brings less burden on CPU resources.

8 Discussion

Communication overhead. Communication overhead is related to the model size. In FL, this can be a trade-off between accuracy and deployment overhead. Higher accuracy may be obtained from larger models to support federated aggregation process, but a larger model may cause more burden to the gateway, which may lead to more model data being shared that would consume the bandwidth resource as well as affect the normal function in gateways.

Privacy preserving. Sharing the local traffic knowledge as model parameters instead of the original telemetry data preserves data privacy. Considering the trade-off between accuracy and privacy, further privacy-preserving mechanisms are not added in this work. Further privacy-preserving mechanisms like encrypted communication or advanced Differential Privacy can be added to prevent malicious interception. But encryption may bring extra overhead to the gateway with limited resources and advanced Differential Privacy may bring slight degradation in federated accuracy [29].

In-network model for traffic analysis. In Section 6, in-network XGBoost-based traffic analysis was introduced and deployed in a federated manner. The federated training framework can also be combined with other state-of-the-art methods like References [25, 51]. Furthermore, the type of in-network model may also be changed to other models supported by Planter [64].

In-network reaction configuration. FLIP4's in-network reaction supports both dropping and metering of the detected threats as discussed in Section 6 and evaluated in Section 7.4. Configuring dropping or metering as the mitigation method is determined by the operator based on the service demands and whether the security level of the IoT network setup is critical. If the network requires a critical security level, then dropping is configured to prevent any potential issues. Conversely, if a lower security level is acceptable, then metering is chosen. This approach reduces traffic disruptions by false alerts that could impact normal service.

Hardware deployment. In this article, we demonstrate our framework's deployment on resource-limited Raspberry Pi and P4-enabled programmable switch. These devices verify our lightweight design. Other hardware deployments (e.g., SmartNICs [53], FPGA [50]) are also viable, depending on service requirements and use cases. They provide more computing resources or flexible operations, adapting to data-intensive use cases like manufacturing or infotainment. The proposed solution may also be deployed on IoT gateway platforms (e.g., Dell Edge Gateway [7]) to enable deployments in real-world IoT scenarios. However, deploying FLIP4 in the field may raise challenges in ML model maintenance, which could be tackled by model updates [58].

9 Conclusion

In this work, we introduced FLIP4, an FL-based in-network traffic analysis solution for distributed gateways. It employs in-network ensemble tree models with federated capabilities for lightweight and privacy-preserving IoT edge deployment. The solution enables fast and accurate traffic analysis within distributed local edge gateways. It shares central knowledge of traffic based on model aggregation without directly sharing the source data. Evaluation on three public datasets demonstrates accurate and fast in-network analysis, outperforming SOTA BNN-based solutions

with lower overheads. FLIP4 ensures a flexible, privacy-preserving design, enabling low-latency response such as anomaly mitigation within resource-constrained gateways on IoT edge.

References

[1] 3GPP. 2019. *Study on Application Architecture for Enabling Edge Applications 3GPP TR 23.758*. 3rd Generation Partnership Project (3GPP).

[2] Albert Gran Alcoz, Martin Strohmeier, Vincent Lenders, and Laurent Vanbever. 2022. Aggregate-based congestion control for pulse-wave DDoS defense. In *Proceedings of the ACM SIGCOMM Conference (SIGCOMM'22)*. Association for Computing Machinery, New York, NY, 693–706. https://doi.org/10.1145/3544216.3544263

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.* 44, 3 (July 2014), 87–95. https://doi.org/10.1145/2656877.2656890

[4] Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (Oct. 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[5] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. 2019. pforest: In-network Inference with Random Forests. Retrieved from https://arXiv:1909.05680.

[6] Enrique Mármol Campos, Pablo Fernández Saura, Aurora González-Vidal, José L. Hernández-Ramos, Jorge Bernal Bernabé, Gianmarco Baldini, and Antonio Skarmeta. 2022. Evaluating federated learning for intrusion detection in internet of things: Review and challenges. *Comput. Netw.* 203 (2022), 108661. https://doi.org/10.1016/j.comnet.2021.108661

[7] DELL Technologies. 2023. Dell EMC Edge Gateway 5200 Software User's Guide. Retrieved from https://dl.dell.com/content/manual77941191-dell-emc-edge-gateway-5200-software-user-s-guide

[8] Yuri Diogenes and Erdal Ozkaya. 2019. *Cybersecurity–Attack and Defense Strategies: Counter Modern Threats and Employ State-of-the-art Tools and Techniques to Protect your Organization Against Cybercriminals*. Packt Publishing Ltd.

[9] Cynthia Dwork. 2006. Differential privacy. In *Automata, Languages and Programming*. Springer, Berlin, 1–12.

[10] ETSI TS 122 261 2021. *Service Requirements for the 5G System (3GPP TS 22.261 Version 16.14.0 Release 16)*. 3rd Generation Partnership Project (3GPP).

[11] Angelo Feraudo et al. 2020. CoLearn: Enabling federated learning in MUD-compliant IoT edge networks. In *Proceedings of the 3rd ACM International Workshop on Edge Systems, Analytics and Networking (EdgeSys'20)*. Association for Computing Machinery, New York, NY, 25–30.

[12] Roy Friedman, Or Goaz, and Ori Rottenstreich. 2021. Clustreams: Data plane clustering. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR'21)*. 101–107.

[13] Nada GabAllah, Ibrahim Farrag, Ramy Khalil, Hossam Sharara, and Tamer ElBatt. 2023. IoT systems with multi-tier, distributed intelligence: From architecture to prototype. *Pervas. Mobile Comput.* 93 (2023), 101818. https://doi.org/10.1016/j.pmcj.2023.101818

[14] Irene Giacomelli, Somesh Jha, Marc Joye, C. David Page, and Kyonghwan Yoon. 2018. Privacy-preserving ridge regression with only linearly-homomorphic encryption. In *Applied Cryptography and Network Security*, Bart Preneel and Frederik Vercauteren (Eds.). Springer International Publishing, Cham, 243–261.

[15] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data? In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS'22)*.

[16] Vladimir Gurevich and Andy Fingerhut. 2021. P4-16 programming for Intel Tofino using intel P4 studio. In *Proceedings of the P4 Workshop*.

[17] Noy Hadar, Shachar Siboni, and Yuval Elovici. 2017. A lightweight vulnerability mitigation framework for IoT devices. In *Proceedings of the Workshop on CPS & IoT Security and Privacy (IoTS&P'17)*. Association for Computing Machinery, New York, NY, 71–75.

[18] Xinhong Hei, Xinyue Yin, Yichuan Wang, Ju Ren, and Lei Zhu. 2020. A trusted feature aggregator federated learning for distributed malicious attack detection. *Comput. Secur.* 99 (2020), 102033. https://doi.org/10.1016/j.cose.2020.102033

[19] Xinpeng Hong, Changgang Zheng, Stefan Zohren, and Noa Zilberman. 2022. Linnet: Limit order books within switches. In *Proceedings of the SIGCOMM Poster and Demo Sessions (SIGCOMM'22)*. 37–39.

[20] Intel. 2021. P4_16 Intel Tofino Native Architecture—Public Version. Retrieved from https://github.com/barefootnetworks/Open-Tofino/blob/master/PUBLIC_Tofino-Native-Arch.pdf

[21] Nicholas Kolokotronis, Maria Dareioti, Stavros Shiaeles, and Emanuele Bellini. 2022. An intelligent platform for threat assessment and cyber-attack mitigation in IoMT ecosystems. In *Proceedings of the IEEE Globecom Workshops (GC'22)*. 541–546. https://doi.org/10.1109/GCWkshps56602.2022.10008548

[22] Arash Habibi Lashkari, Andi Fitriah A. Kadir, Laya Taheri, and Ali A. Ghorbani. 2018. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *Proceedings of the International Carnahan Conference on Security Technology (ICCST'18)*. 1–7.

[23] Jong Hyouk Lee and Kamal Singh. 2020. SwitchTree: In-network computing and traffic analyses with random forests. *Neural Comput. Appl.* 32, 1 (2020), 1–12.

[24] Jianhua Li, Lingjuan Lyu, Ximeng Liu, Xuyun Zhang, and Xixiang Lyu. 2022. FLEAM: A federated learning empowered architecture to mitigate DDoS in industrial IoT. *IEEE Trans. Industr. Info.* 18, 6 (2022), 4059–4068. https://doi.org/10.1109/TII.2021.3088938

[25] Q. Li, Zeyi Wen, and Bingsheng He. 2019. Practical federated gradient boosting decision trees. Retrieved from https://abs/1911.04206

[26] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. 2023. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Trans. Knowl. Data Eng.* 35, 4 (Apr. 2023), 3347–3366.

[27] Yi Liu, James J. Q. Yu, Jiawen Kang, Dusit Niyato, and Shuyu Zhang. 2020. Privacy-preserving traffic flow prediction: A federated learning approach. *IEEE Internet Things J.* 7, 8 (2020), 7751–7763. https://doi.org/10.1109/JIOT.2020.2991401

[28] Heiko Ludwig et al. 2020. IBM Federated Learning: An Enterprise Framework White Paper V0.1. Retrieved from https://arXiv:2007.10987

[29] Samuel Maddock, Graham Cormode, Tianhao Wang, Carsten Maple, and Somesh Jha. 2022. Federated boosted decision trees with differential privacy. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'22)*.

[30] H. Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated Learning of Deep Networks using Model Averaging. Retrieved from https://arXiv:1602.05629

[31] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N. Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. 2017. IoT sentinel: Automated device-type identification for security enforcement in IoT. In *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems (ICDCS'17)*. 2177–2184.

[32] Ilya Mironov. 2017. Rényi differential privacy. In *Proceedings of the IEEE 30th Computer Security Foundations Symposium (CSF'17)*. 263–275.

[33] Thien Duc Nguyen, Samuel Marchal, Markus Miettinen, Hossein Fereidooni, N. Asokan, and Ahmad-Reza Sadeghi. 2018. DÏoT: A federated self-learning anomaly detection system for IoT. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'18)*. 756–767.

[34] Council of European Union. 2016. General Data Protection Regulation. Retrieved from https://https://gdpr-infor.eu

[35] Office of the California Attorney General. 2020. California Consumer Privacy Act (CCPA): First Modified Regulations. Retrieved from https://oag.ca.gov/sites/all/fles/agweb/pdfs/privacy/ccpa-text-of-mod-clean-020720.pdf

[36] National People's Congress of the People's Republic of China. 2021. Personal Information Protection Law of the People's Republic of China. Retrieved from https://personalinformationprotectionlaw.com/

[37] Oladayo Olufemi Olakanmi and Kehinde Oluwasesan Odeyemi. 2021. Throttle: An efficient approach to mitigate distributed denial of service attacks on software-defined networks. *Secur. Privacy* 4, 4 (2021), e158. https://doi.org/10.1002/spy2.158

[38] Xinjun Pei, Xiaoheng Deng, Shengwei Tian, Lan Zhang, and Kaiping Xue. 2023. A knowledge transfer-based semi-supervised federated learning for IoT malware detection. *IEEE Trans. Depend. Secure Comput.* 20, 3 (2023), 2127–2143.

[39] Larry L. Peterson, Carmelo Cascone, Brian O'Connor, Thomas Vachuska, and Bruce Davie. 2021. *Software-defined Networks: A Systems Approach*. Systems Approach LLC.

[40] Qiaofeng Qin, Konstantinos Poularakis, Kin K. Leung, and Leandros Tassiulas. 2020. Line-speed and scalable intrusion detection at the network edge via federated learning. In *IFIP Networking*. IEEE, 352–360.

[41] Pethuru Raj, Kavita Saini, and Chellammal Surianarayanan. 2022. *Edge/Fog Computing Paradigm: The Concept, Platforms and Applications*. Academic Press.

[42] Habib F. Rashvand, Khaled Salah, Jose M. Alcaraz Calero, and Lein Harn. 2010. Distributed security for multi-agent systems–review and applications. *IET Info. Secur.* 4, 4 (2010), 188–201.

[43] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. 2023. ELSA: Secure aggregation for federated learning with malicious actors. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'23)*. 1961–1979.

[44] Sumudu Samarakoon, Mehdi Bennis, Walid Saad, and Mérouane Debbah. 2020. Distributed federated learning for ultra-reliable low-latency vehicular communications. *IEEE Trans. Commun.* 68, 2 (2020), 1146–1159.

[45] Osama Shahid, Viraaji Mothukuri, Seyedamin Pouriyeh, Reza M. Parizi, and Hossain Shahriar. 2021. Detecting network attacks using federated learning for IoT devices. In *Proceedings of the IEEE 29th International Conference on Network Protocols (ICNP'21)*. 1–6. https://doi.org/10.1109/ICNP52444.2021.9651915

[46] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP'18)*. 108–116.

[47] Giuseppe Siracusano et al. 2022. Re-architecting traffic analysis with neural network interface cards. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*. 513–533.

[48] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. 2021. Open for hire: Attack trends and misconfiguration pitfalls of IoT devices. In *Proceedings of the 21st ACM Internet Measurement Conference*. New York, NY, 195–215. https://doi.org/10.1145/3487552.3487833

[49] Radostin Stoyanov, Adam Wolnikowski, Robert Soulé, Sándor Laki, and Noa Zilberman. 2022. Building an internet router with P4Pi. In *Proceedings of the ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS'22)*. 151–156.

[50] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. 2022. Taurus: A data plane architecture for per-packet ML. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*. Association for Computing Machinery, New York, NY, 1099–1114.

[51] Zhihua Tian, Rui Zhang, Xiaoyang Hou, Lingjuan Lyu, Tianyi Zhang, Jian Liu, and Kui Ren. 2024. FederBoost: Private federated learning for GBDT. *IEEE Trans. Depend. Secure Comput.* 21, 3 (2024), 1274–1285. https://doi.org/10.1109/TDSC.2023.3276365

[52] Han Wang, Luis Muñoz González, David Eklund, and Shahid Raza. 2021. Non-IID data re-balancing at IoT Edge with peer-to-peer federated learning for anomaly detection. In *Proceedings of the 14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'21)*. Association for Computing Machinery, New York, NY, 153–163. https://doi.org/10.1145/3448300.3467827

[53] Bruno Missi Xavier, Rafael Silva Guimarães, Giovanni Comarela, and Magnos Martinello. 2022. MAP4: A pragmatic framework for in-network machine learning traffic classification. *IEEE Trans. Netw. Serv. Manage.* 19, 4 (2022), 4176–4188.

[54] Bin Xiao, Wei Chen, and Yanxiang He. 2008. An autonomous defense against SYN flooding attacks: Detect and throttle attacks at the victim side independently. *J. Parallel Distrib. Comput.* 68, 4 (2008), 456–470. https://doi.org/10.1016/j.jpdc.2007.06.013

[55] Shengjie Xu, Yi Qian, and Rose Qingyang Hu. 2019. Data-driven network intelligence for anomaly detection. *IEEE Netw.* 33, 3 (2019), 88–95. https://doi.org/10.1109/MNET.2019.1800358

[56] Li Yang, Abdallah Moubayed, Ismail Hamieh, and Abdallah Shami. 2019. Tree-based intelligent intrusion detection system in internet of vehicles. In *Proceedings of the IEEE Global Communications Conference (GLOBECOM'19)*. 1–6.

[57] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: concept and applications. *ACM Trans. Intell. Syst. Technol.* 10, 2, Article 12 (Jan. 2019), 19 pages. https://doi.org/10.1145/3298981

[58] Mingyuan Zang, Changgang Zheng, Lars Dittmann, and Noa Zilberman. 2023. Towards continuous threat defense: In-network traffic analysis for IoT gateways. *IEEE Internet Things J.* 11, 6 (2023), 9244–9257. https://doi.org/10.1109/JIOT.2023.3323771

[59] Tuo Zhang, Chaoyang He, Tianhao Ma, Lei Gao, Mark Ma, and Salman Avestimehr. 2021. Federated learning for internet of things. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems (SenSys'21)*. Association for Computing Machinery, New York, NY, 413–419. https://doi.org/10.1145/3485730.3493444

[60] Changgang Zheng, Xinpeng Hong, Damu Ding, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. 2023. In-network machine learning using programmable network devices: A survey. *IEEE Commun. Surveys Tutor.* 26, 2 (2023), 1171–1200. https://doi.org/10.1109/COMST.2023.3344351

[61] Changgang Zheng, Benjamin Rienecker, and Noa Zilberman. 2023. QCMP: Load balancing via in-network reinforcement learning. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Future of Internet Routing & Addressing*.

[62] Changgang Zheng, Haoyue Tang, Mingyuan Zang, Xinpeng Hong, Aosong Feng, Leandros Tassiulas, and Noa Zilberman. 2023. DINC: Toward distributed in-network computing. *Proc. ACM Netw.* 1, CoNEXT3, Article 14 (Nov. 2023), 25 pages. https://doi.org/10.1145/3629136

[63] Changgang Zheng, Zhaoqi Xiong, Thanh T. Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. 2024. IIsy: Hybrid in-network classification using programmable switches. *IEEE/ACM Trans. Netw.* 32, 3 (2024), 2555–2570.

[64] Changgang Zheng, Mingyuan Zang, Xinpeng Hong, Liam Perreault, Riyad Bensoussane, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. 2024. Planter: Rapid prototyping of in-network machine learning inference. *ACM SIGCOMM Comput. Commun. Rev.* 54, 1 (2024), 2–21.

[65] Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. 2023. An efficient design of intelligent network data plane. In *Proceedings of the USENIX Security Symposium*.

Received 15 January 2024; revised 22 July 2024; accepted 21 August 2024