# Towards Continuous Threat Defense: In-Network Traffic Analysis for IoT Gateways

Mingyuan Zang, Changgang Zheng, Lars Dittmann, and Noa Zilberman

***Abstract*— The widespread use of IoT devices has unveiled overlooked security risks. With the advent of ultra-reliable low-latency communications (URLLC) in 5G, fast threat defense is critical to minimize damage from attacks. IoT gateways, equipped with wireless/wired interfaces, serve as vital frontline defense against emerging threats on IoT edge. However, current gateways struggle with dynamic IoT traffic and have limited defense capabilities against attacks with changing patterns. In-network computing offers fast machine learning-based attack detection and mitigation within network devices, but leveraging its capability in IoT gateways requires new continuous learning capability and runtime model updates. In this work, we present P4Pir, a novel in-network traffic analysis framework for IoT gateways. P4Pir incorporates programmable data plane into IoT gateway, pioneering the utilization of in-network machine learning (ML) inference for fast mitigation. It facilitates continuous and seamless updates of in-network inference models within gateways. P4Pir is prototyped in P4 language on Raspberry Pi and Dell Edge Gateway. With ML inference offloaded to gateway's data plane, P4Pir's in-network approach achieves swift attack mitigation and lightweight deployment compared to prior ML-based solutions. Evaluation results using three public datasets show that P4Pir accurately detects and fastly mitigates emerging attacks (>30% accuracy improvement and sub-millisecond mitigation time). The proposed model updates method allows seamless runtime updates without disrupting network traffic.**

***Index Terms*—In-network computing; Machine learning; Security; Internet of Things; P4**

## I. Introduction

The ubiquitous and dynamic deployment of IoT devices in diverse use case environments has given rise to a surge in security threats that were previously overlooked. The threats become more urgent in the context of ultra-reliable low-latency communications (URLLC) in 5G [1]. With end-to-end latency at millisecond scale under URLLC requirements, threats can spread faster than cloud-based security analytics can respond. This is particularly pressing in time-critical applications such as real-time sensing (latency $< 30ms$) and industrial process automation (latency $< 50ms$) [2].

Threats are also evolving stealthily with changing patterns. Attackers exploit botnets or alter attack patterns [3] to evade traditional security measures like firewalls, causing disruption of critical services or impact on network infrastructure. Thereby, swift defenses against evolving attacks are imperative to counteract potential widespread damage. IoT gateways (with wireless/wired interface) close to end devices play a vital role in defending against emerging threats before they spread. However, to counter emerging attacks in gateway, state-of-the-art security measures focus primarily on accurate detection through methods like online learning or model retraining [4, 5]. This leaves a gap in fast and flexible action enforcement against detected anomalies.

Recent advancements in programmable data planes (PDP) and in-network machine learning (ML) inference provide a novel approach to **rapidly detect attacks and enforce actions** in high-speed switches [6, 7], with potential extending to other network devices like IoT gateways. In-network ML inference, a type of in-network computing, is the offloading of ML inference tasks from servers/cloud to the data plane of network devices. Unlike classical practices of running ML training and inference solely in servers/cloud, this approach offloads inference processes from servers/cloud to pipeline logics in programmable data plane within network devices (programmed in P4 language [8]). Thus, ML inference runs concurrently with packet switching. This empowers a direct ML-based identification of malicious traffic and action enforcement within network devices, avoiding Round-Trip Times (RTT) for cloud-based analysis [6]. Moreover, reconfigurable pipelines in programmable data planes also enable **flexible defense** in network devices. Traffic features can be flexibly parsed and in-network ML inference model can be reconfigured to flexibly respond to emerging threats, unlike classical solutions where ML models rely on predefined features from either monitoring protocols [9] or proprietary toolkits [10].

However, there are still challenges when it comes to integrating the **efficiency and flexibility** of in-network ML to enable fast and continuous attack detection and mitigation within IoT gateway. First, it poses a challenge to integrate in-network ML inference into the resource-constrained gateway device. Second, there is a need for continuous collection and analysis of traffic features to learn emerging threats [11]. Third, IoT gateways need to operate continuously to minimize maintenance costs, and ensure uninterrupted service delivery along with round-the-clock security [12]. This raises a concern: reconfiguring in-network ML functions requires data plane program recompilation, potentially disrupting normal traffic in gateway during updates. To address these challenges, the question is: *Can in-network ML be used to detect and mitigate emerging security threats at the IoT edge, while providing high detection accuracy, fast mitigation and continuous learning without affecting normal traffic?*

To answer this question, we present P4Pir, a novel in-network ML-based analysis framework providing swift ML-

This work was partly supported by the Otto Mønsted Foundation, VMware, and EU Horizon SMARTEDGE (101092908).

M. Zang and L. Dittmann are with Technical University of Denmark, Denmark. (email: minza@dtu.dk, ladit@fotonik.dtu.dk)

C. Zheng and N. Zilberman are with University of Oxford, UK. (email: {name.surname}@eng.ox.ac.uk)

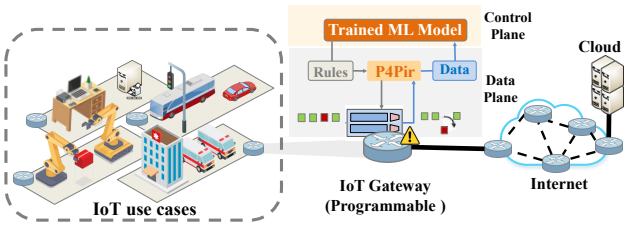



Fig. 1. Deploying P4Pir within IoT gateway in edge scenario.

based attack defense, continuous traffic learning, and seamless model updates in IoT gateway device (shown in Figure 1). P4Pir operates within an IoT gateway with programmable data plane on network edge. P4Pir establishes a seamless workflow for collecting traffic data, updating an in-network ML model, and employing data plane table rules to detect/mitigate threats within the gateway. Specifically, P4Pir incorporates the following three features: *a) Swift in-network ML-based mitigation.* We integrate in-network ML inference into an IoT gateway, supporting fast threat defense as incoming traffic passes through the pipeline. *b) Continuous learning with proactive logging and labeling.* We introduce proactive traffic data logging and automated labeling to avoid manual intervention, enabling continuous retraining of the in-network inference model and learning on emerging attacks. *c) Seamless updates of in-network inference model.* We propose shadow table updates scheme to allow seamless in-network model updates to identify emerging attacks, avoiding function recompilation or forwarding disruptions in gateway.

To the best of the authors' knowledge, P4Pir is the first work offering in-network ML in IoT gateway. Additionally, P4Pir is the first to explore a seamless reconfiguration of in-network ML inference at runtime within IoT gateway. This allows fast and continuous defense against emerging attacks, ensuring uninterrupted service.

Our contributions are as follows:

- An in-network traffic analysis framework in IoT gateway, leveraging in-network ML inference for accurate detection and fast mitigation of emerging attacks.
- A proactive logging and unsupervised labeling to continuously log traffic features and learn from incoming traffic.
- A seamless reconfiguration method for in-network ML inference at runtime, allowing seamless model updates for continuous defense without interrupting gateway service.
- A prototype[1] on both P4Pi (a Raspberry Pi-based platform providing data plane programmability) and Dell Edge Gateway. Evaluation results show that P4Pir can efficiently detect and fastly mitigate emerging attacks (>30% accuracy enhancement and sub-millisecond mitigation time) with negligible overhead. Compared with Kitsune [4], a state-of-the-art IoT gateway intrusion detection system, P4Pir achieves equal or 10% higher detection accuracy and enables fast attack mitigation.

The remainder of the paper is organized as follows: Section II provides background to IoT gateways and programmable data planes. Section III discusses related work and limitations. Section IV provides an overview of the proposed solution, and design details are explained in Section V-VIII. Section IX presents a comprehensive evaluation and Section X compares P4Pir with state-of-the-art work. Section XI provides a discussion and Section XII concludes.

[1]Source code: https://github.com/In-Network-Machine-Learning/P4Pir

## II. Background

### A. Traffic Analysis on IoT Gateway

An IoT gateway acts as a bridge between local network devices (sensors, actuators, etc.) and remote servers or cloud platforms. It has wireless interface that connects to local devices to collect user data and wired interface that routes the data to the Internet [13]. Deployment scenario's requirements determine the selection of communication protocols, such as DNS or HTTP for service communication and MQTT for sensor data collection and cloud analytics [14]. For specific use cases like industrial automation, protocols like ModBus/TCP [15] are configured to enable ModBus interface support.

Various protocols have security concerns that attackers exploit. Low-layer packet headers remain vulnerable even after encryption [16]. Traditional solutions for traffic analysis in IoT gateways may lead to false positive alerts and limited flexibility in access control and filtering [17, 18]. Deploying machine learning algorithms in the cloud [19, 20] for training and inference improves detection accuracy but introduces delays in mitigation decisions [6]. Swift reactions are essential for low-latency URLLC services in 5G, as delayed mitigation or false detection may cause widespread attacks and heavily impact the network infrastructure [21]. A key challenge yet to be tackled is efficiently offloading ML inference to IoT gateways, ensuring accurate and swift attack mitigation.

### B. Programmable Data Planes

While Software-defined Networking (SDN) centralized the control plane to allow flexible control over network devices [22], the advent of Protocol-Independent Switch Architecture (PISA) and programming protocol-independent packet processors (P4) [8] further equip the data plane with programmability and flexibility. Figure 2 presents a schematic diagram of the PISA architecture for programmable data plane. It includes three main components: parser, reconfigurable Match-Action (M/A) pipeline, and deparser. P4 language defines how incoming packets are processed in such architecture. Programmable Data Plane enables programmability in the

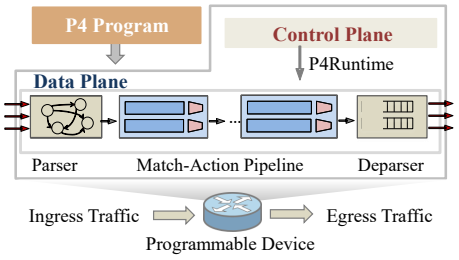



Fig. 2. Schematic PISA architecture for programmable data plane in a programmable device (switch) [22]. It can be programmed by P4 language.

TABLE I
RELATED WORKS OF ML-BASED ATTACK DETECTION

| Reference | Feature Collection | Detection Algorithm† | Inference Location | Detection | Mitigation | Runtime Update | IoT Deployment |
|---|---|---|---|---|---|---|---|
| PassbanIDS[30] | Dumped pcap | LOF/iForest | Gateway control plane | ✓ | ✗ | ✗ | ✓ |
| Kitsune[4] | Dumped pcap | AE | Gateway control plane | ✓ | ✗ | ✓* | ✓ |
| Qin[31] | In-band | NN | SDN controller | ✓ | ✗ | ✗ | ✓ |
| Musumeci[32] | In-band | RF, kNN, SVM | SDN controller | ✓ | ✗ | ✗ | ✗ |
| SwitchTree[6] | In-band | DT | Switch data plane | ✓ | ✓ | ✗ | ✗ |
| P4Pir | In-band | DT, RF | Gateway data plane | ✓ | ✓ | ✓ | ✓ |

† RF - Random Forest, NB - Naïve Bayes, LSTM - Long Short-Term Memory, LOF - Local Outlier Factor, iForest - Isolation Forest, AE - AutoEncoder, NN - Neural Network, kNN - k-Nearest Neighbor, SVM - Support Vector Machine
* Online learning

following aspects: a) Packet header parsing: protocol identification and header extraction; b) Packet processing: packet manipulation with reconfigurable M/A tables in pipelines; c) Packet header deparser: packet header reconstruction. The control plane can access the data plane and CPU processor. Once a P4 program is compiled and run on a progammable device, the control plane can reconfigure the data plane at runtime via P4Runtime interface [23].

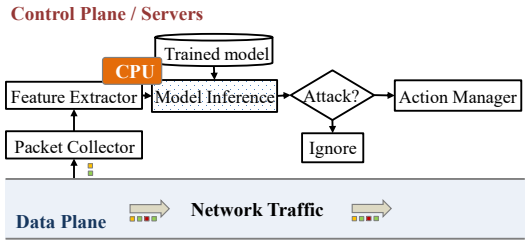
Programmable data planes enable in-network computing and in-network ML, offloading applications like ML inference from the server to the data plane [6]. This allows ML-based inference for classification tasks to be performed directly in the data plane, analyzing incoming traffic using ML models without involving the server/cloud/SDN controller. This approach offers advantages in traffic feature analysis and enables quick decision-making. In-network ML has been applied in traffic classification [24], attack detection [6, 25], elephant flow prediction [26, 27], and time-series financial data prediction [28, 29]. In terms of IoT gateways, however, it remains unclear how these benefits can be effectively leveraged.

## III. RELATED WORK AND GAPS

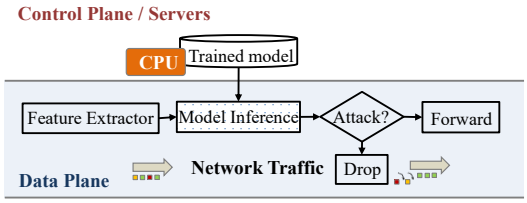
Table I lists a summary of related work. The literature is summarized in three aspects: ML deployment positions, feature collection, and update mechanisms.

In a traditional or SDN-enabled IoT network, ML model training and inference are both deployed on the server or controller [30, 33, 34]. ML inference solutions rely on processor-based ML frameworks, where traffic needs to be sent from the data-plane-pipeline to the processor (e.g. CPU/GPU) for inference decision, bringing extra overheads and increased latency for attack mitigation (as Figure 3 (a)). In-network ML deployment can reduce such inference latency by offloading the inference process from the controller. Figure 3 (b) depicts how inference model can be offloaded to data plane together with traffic forwarding logic. Prior work like SwitchTree [6] has applied such an approach for fast attack detection. By training a model offline and mapping the trained inference model to the data plane, malicious traffic can be labeled and mitigated directly based on in-network inference results. Despite accurate detection and swift mitigation, such work requires P4 program recompilation instead of runtime updates to achieve continuous learning, interrupting packet forwarding.

When it comes to feature collection to support continuous learning, depending on model inference location, collection




(a) Traditional ML deployment: inference on control plane/server.




(b) In-network ML deployment: inference on data plane.

Fig. 3. (a) Traditional [30] vs. (b) In-network ML deployment used in this work for traffic analysis and attack detection.

methods in prior work can be classified into two types: offline collection and online collection. In offline methods, features are extracted from dumped traces so that model can be trained and evaluated in a clean environment [4]. In SDN-enabled gateways, a controller collects traffic statistics via OpenFlow interface [34]. When programmable data plane is introduced, in-band feature collection has been applied to flexibly parse features upon packet arrival [32, 35, 36]. This approach is well-suited for IoT scenarios [37, 38]. It also enables the direct feeding of collected features to in-network ML model.

Continuous learning of ML models is necessary to address potential data drift, change of feature distribution or emerging malicious activities [39]. Researchers have studied using online learning to continuously learn from collected traffic pattern, but is vulnerable to data distribution [4]. Alternatively, the deployed supervised-based model can be retrained and redeployed periodically to accommodate new traffic patterns [31]. Despite its reliable detection performance from supervised learning, this method needs model reloading and may lead to service disruption, especially in the in-network ML deployment. Regarding this in-network scenario, model updates possibility is discussed in [25]. However, there lack of further investigation into continuous learning approach and seamless model updates at runtime for IoT gateway deploy-

ment to minimize service disruption during model updates.

### A. Limitations of Current Solutions

While ML has been studied in multiple works for traffic analysis and attack detection at IoT gateway, limitations still exist in continuous ML model update and fast attack mitigation. Despite model deployment on server/SDN controller allowing for easy maintenance and updates, it lacks timely attack mitigation at the gateway due to traversal time. Conversely, in-network ML inference provides fast attack mitigation but lacks efficient runtime model updates, as compile-time updates for in-network inference models can disrupt traffic forwarding. To fill the gap, P4Pir integrates in-network ML inference in IoT gateway and incorporates techniques to support timely ML-based attack mitigation, as well as runtime updates for continuous and seamless model maintenance.

## IV. System Design

### A. Threat Model

This work focuses on attacks leveraging network protocols, specifically IoT edge deployment scenarios involving gateway connections (referred to as the neighbor and tenant scenarios as described by Wang et al. [40]). In the neighbor scenario, attackers connect to the same network as victim devices and launch attacks through the network connection. In the tenant scenario, attackers gain access to rented IoT devices using obtained credentials, exploiting them as botnets for further attacks. By exploiting network vulnerabilities for attack chain, attackers can carry out the following types of attacks:

- Passive attacks: Attackers perform actions like scanning to gather network information, including vulnerability scanning for potential weaknesses and port scanning to identify open ports on victims' devices.
- Active attacks: Attackers actively exploit end devices as botnets to flood packets to the server by initiating massive amounts of requests to cause Denial of Service/Distributed Denial of Service (DoS/DDoS). Such malicious traffic is more volumetric in terms of packet requests or anomalous retransmissions.

To effectively counteract these attacks and prevent them from impact on other parts of network, it is crucial to address the ever-changing patterns (e.g. pulse-wave DDoS attacks [3]) of malicious behavior and provide early mitigation. In this study, we specifically consider scenarios where other defense mechanisms are bypassed by attackers or not in place, and the IoT gateway serves as the first line of defense [41].

**Our main goal** is to swiftly mitigate diverse and emerging types of attacks in IoT environments by leveraging and continuously updating in-network ML inference within IoT gateway. Thereby, we can enable rapid response to detected attack chains, minimizing the impact of attackers bypassing traditional security measures.

### B. Design Overview

P4Pir uses in-network ML inference to detect and mitigate attacks at the gateway. Compared with existing in-network inference solutions, running entirely in the data plane

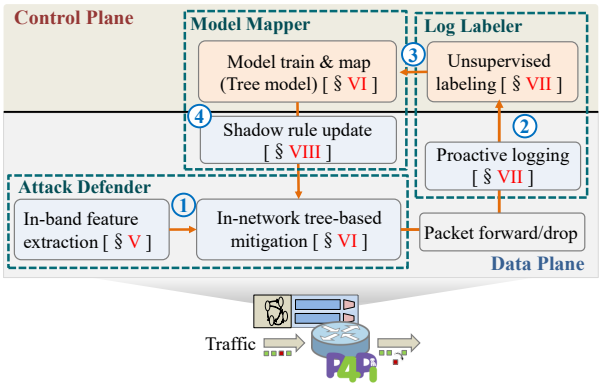



Fig. 4. Design overview of framework in P4Pir.

(e.g., [6, 25]), P4Pir engages continuous model updates at runtime without interrupting the traffic processing in the gateway. Figure 4 depicts P4Pir's framework. It consists of three main components containing a series of function blocks enabling swift ML-based attack defense with uninterrupted model updates in the IoT gateway.

- **Attack defender** performs ML-based attack detection and fast mitigation within the data plane. This defense mechanism involves two function blocks: in-band feature extraction (§V) and in-network tree-based attack mitigation (§VI). First, relevant traffic features are extracted as inputs to an in-network model. Next, an in-network inference tree model, which is mapped to data plane, identifies malicious traffic based on these features (Figure 4 step ①). As a result, benign traffic is allowed to pass through, while detected malicious traffic is swiftly dropped, effectively mitigating the potential threat.
- **Log labeler** automates a proactive logging process from the data plane to the control plane. It has two function blocks: proactive logging and unsupervised labeling (§VII). During proactive logging, extracted features are promptly logged to the control plane (Figure 4 step ②). At the control plane, these logs are labeled using the unsupervised algorithm *iForest*. This approach allows for continuous learning of incoming traffic patterns.
- **Model mapper** serves the purpose of training and mapping the in-network model (Figure 4 step ③) Its main function is to generate and seamlessly update Match-Action table entries, allowing for the runtime configuration of in-network inference. It involves two function blocks: model train & map (§VI), shadow rule updates (§VIII). The "Model Train & Map" block serves two purposes. Firstly, it retrains the model based on logs, allowing updates to adapt to new traffic patterns. Secondly, new table rules are generated to map the retrained model's parameters. Next, the "Shadow rule update" block inserts these new rules into the data plane using a shadow update method (as shown in Figure 4 step ④). This seamless configuration of in-network ML inference ensures uninterrupted traffic forwarding in the data plane.

With this framework design, our solution continuously learns from newly incoming traffic and fastly mitigates abnormal traffic. In the following sections, we provide a detailed explanation of the internal design of each block as shown in Figure 5.

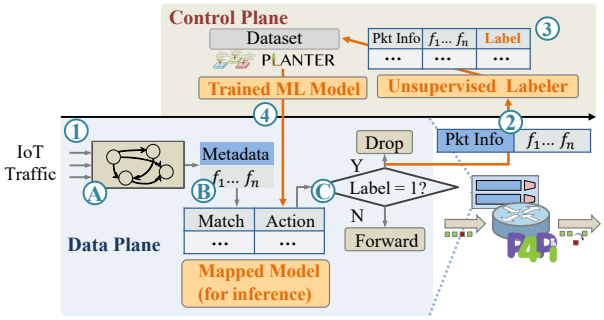



Fig. 5. Detailed workflow of in-network ML-based traffic analysis in P4Pir.

## V. In-Network Feature Extraction

This "In-network feature extraction" block enables real-time feature extraction within IoT gateway's data plane, where programmable data plane supports flexible IoT traffic parsing. Important features are identified through offline analysis to determine which ones should be extracted from incoming traffic. Extracted features are then used for in-network ML inference in data plane and logging toward control plane.

### A. Offline Feature Analysis

Offline feature analysis is conducted to determine relevant features based on their importance. It examines the varying weights of features in revealing malicious events. The identified relevant features are used for ML-based detection. There are several methods to calculate the feature importance. In this paper, we use *Permutation Importance*. Compared to impurity-based feature importance algorithm [42], widely used in tree-based ML research, *Permutation Importance* is suitable for tabular data like network traffic, where feature importance is computed by the degree to which the model performance score decreases after the feature is randomly shuffled [43]. The computation of permutation importance is written as:

$$i_j = s - \frac{1}{K}\sum_{k=1}^{K} s_{k,j}, \tag{1}$$

where $s$ is a reference score computed from the accuracy of model $m$ on Data $D$. To compute the importance score $i_j$ of feature $j$ in the shuffled data, this process repeats $K$ time. By doing so, the bias on cardinality features, like the numerical ones, can be reduced [44].

We select features for in-band feature extraction within the data plane by considering the analysis results of feature importance and the feasibility of in-band extraction. Public dataset *EDGE-IIOTSET* [45] is used as an example, and the selected features are listed in Table II.

### B. In-Band Feature Extraction

In P4Pir, selected features (as listed in Table II) are extracted from incoming traffic within IoT gateway in an in-band manner. This in-band extraction method (Figure 5 Step 1) offers flexibility and immediacy by directly extracting features from different layers in the data plane, as opposed to the traditional method of extracting features from dumped captures.

In-band feature extraction is achieved through protocol-independent processing pipelines in the programmable data plane, which programmatically process packet headers. P4 language [8] defines the packet header fields, allowing the parser to identify protocols and extract the header information (Figure 5 step 1-A). The parser operates as a state machine, parsing headers layer by layer (Figure 6). State transitions occur based on identifiers in packet headers, such as EtherType in Ethernet header indicating an IPv4 packet (*0x0800*). Transport-layer packets are identified using the protocol field in IP header, while application-layer packets are recognized by destination port number (assuming default port configurations). If a header field indicates the next header in an upper layer, the state transits to extract features from that header. This parser design can be programmed in P4 to support various protocol definitions, enabling flexible in-band packet processing.

When state transition reaches an end and all $N$ features $[f_1, f_2, ..., f_n]$ are extracted from the header field, they are temporarily saved in metadata (Figure 5 step 1-B).

TABLE II
List of Selected Traffic Features.

| Layer | Features |
|---|---|
| Network Layer | Source/destination IP addresses |
| Transport Layer | Source/destination ports, TCP flags, sequence/acknowledgment number, length. |
| Application Layer | MQTT protocol length, MQTT version, DNS query type, Modbus/TCP length, Modbus/TCP unit ID. |

## VI. In-Network Tree-Based Mitigation

This "In-network tree-based mitigation" block conducts in-network tree-based inference on extracted traffic features to detect and mitigate malicious traffic, preventing its forwarding to the next hop and minimizing the impact on network (Figure 5 1-C).

In this section, we explain in more detail the concept of in-network inference and the process of training and mapping a tree-based ML model to the data plane using Planter [46]. This involves training the model and obtaining its parameters, followed by a model mapping process that translates the

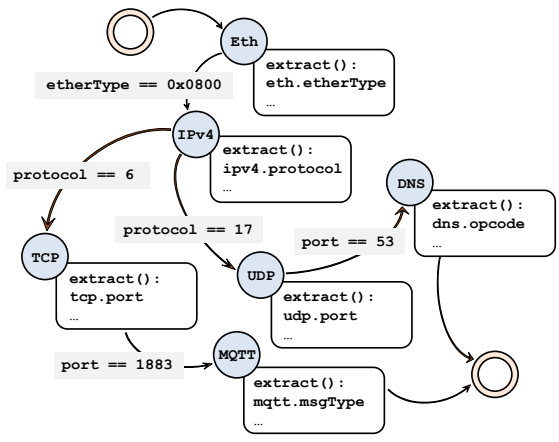



Fig. 6. Flow chart of a part of parser's state transition. Key transition conditions are marked in gray.

trained inference model into executable code for in-network inference within the data plane pipeline. We then explain how in-network inference benefits fast mitigation in P4Pir.

### A. Inference Model Mapping to Match-Action Tables

To ensure optimal performance and scalability in the data plane pipeline [46], our solution leverages tree-based ML models such as Decision Trees (DT) and Random Forest (RF) for in-network inference on tabular traffic features [47]. Tree-based models are preferred over deep learning models because although deep learning methods, like Neural Networks (NN) or Deep Neural Networks (DNN), have shown outstanding accuracy in detection, they are computing-intensive and typically rely on servers or cloud platforms with high processing power rather than IoT gateways at the edge [27, 46]. Therefore, deploying complex deep learning models on resource-limited IoT gateways might not be a cost-effective choice.

To achieve in-network ML inference, ML model training and inference are separated, with training conducted in the control plane and inference performed in the data plane of the gateway. This separation allows for efficient offloading of ML inference to the data plane, enabling fast decision-making and malicious mitigation in an in-network manner. Previous approaches (e.g. [32]) used packages like *scikit-learn* [48] for supervised model training to obtain a model structure and parameters for inference. However, it is challenging to load such inference into the data plane because the pipeline architecture is very different from processors like CPUs. Thereby, in order to process in-network ML inferences within the data plane pipeline, it is necessary to translate the inference model into a set of Match-Action (M/A) tables in the form of PISA architecture [8] (as plotted in Figure 2).

**Challenge: flexible in-network model mapping.** A flexible mapping method is critical to enable runtime reconfigurability and ensure uninterrupted service in IoT gateways. However, existing in-network model mapping solutions are limited in flexibility. Xavier et al. [49] proposed a method where the model is translated into hard-coded *if-else* statements in P4 language, reducing implementation overhead but requiring recompilation for parameter updates. Another solution *pForest* [6, 25] uses M/A tables for flexibility but maintains a sequential dependency among tables, impacting scalability [46].

**Solution: encode-based model mapping method for tree-based model.** To further enhance flexibility and optimize resource scalability, we adopt an encode-based mapping solution proposed in *Planter* [7]. An encode-based mapping solution encodes the parameters in M/A tables to optimize the resource utilization of the data plane. It breaks the dependency within the mapped tree path by encoding the feature space [46].

Figure 7 presents an example DT to illustrate how DT can be encoded and mapped to a set of M/A tables. Figure 7 (a) illustrates a simple DT structure trained with two traffic features $[f_1, f_2]$. The tree performs binary splits based on threshold values for each feature, and inference results are obtained at the leaf nodes. Figure 7 (b) presents the corresponding feature space of the tree, divided into rectangular regions by the thresholds. Previous methods [6] mapped these thresholds to M/A rules by following the sequential top-down tree path, causing extra stage consumption in data plane pipeline. In contrast, the encode-based method proposed in *Planter* [7] effectively breaks the sequential dependency by encoding thresholds into M/A rules in feature tables, as in Figure 7 (c). It means each rectangular area in Figure 7 (b) is encoded to a pair of codes (e.g. $\{f_1 \in [0, th_{11}], f_2 \in [0, th_{22}]\}$ is encoded as *0000*) in feature tables, and a decision table decodes the decision array at the leaf node to produce a binary label output (e.g. packet with features' range in $\{f_1 \in [0, th_{11}], f_2 \in [0, th_{22}]\}$ is determined as *1* - malicious). This parallelizes the pipeline traversal process, allowing for more efficient usage of pipeline resources, offering flexible and low-latency inference.

Expanding this concept, a Random Forest model is encoded as ensemble of multiple trees. In line with [7], each feature table includes coding pairs from all trees, and a voting table synthesizes decision results from these features and trees.

### B. In-Network ML-Based Mitigation

Mapping a trained tree model to M/A tables enables in-network inference, circumventing the need to send incoming traffic features (extracted in-band as explained in Section V) to the control plane or servers. Extracted features traverse these M/A tables in the data plane, akin to traversing the trained tree model, to obtain inference decisions as final *Actions*. The binary values in tables determine packet handling, where 0 represents benign packets and 1 represents malicious ones. This approach allows for swift mitigation by forwarding benign packets and dropping malicious ones within the gateway, effectively halting malicious traffic (Figure 5 step C).

## VII. Proactive Logging and Unsupervised-based Log Labeling

P4Pir goes beyond utilizing extracted features for in-network mitigation decisions; it also provides proactive logging and unsupervised-based labeling of these features, automating the entire logging process. This approach prepares for retraining and seamless updates of the in-network model, without the need for manual intervention from administrators.

### A. Proactive Logging

P4Pir incorporates proactive logging in the data plane, where extracted features are sent to the control plane simultaneously with traffic detection (Figure 5 step 2). This logging enables the control plane to receive records of incoming traffic patterns, facilitating continuous learning of emerging traffic patterns.

**Challenge: efficient proactive logging.** Proactive logging in the data plane has the potential increase in traffic volume caused by per-packet processing during feature extraction. Given the limited resources in the IoT gateway, efficient logging with low overhead is mandatory.

**Solution: digest-based logging.** To select low-overhead logging method, two commonly used P4-based messaging solutions are compared: direct transmission of the entire packet as *packet_in* to the control plane via the CPU port, versus

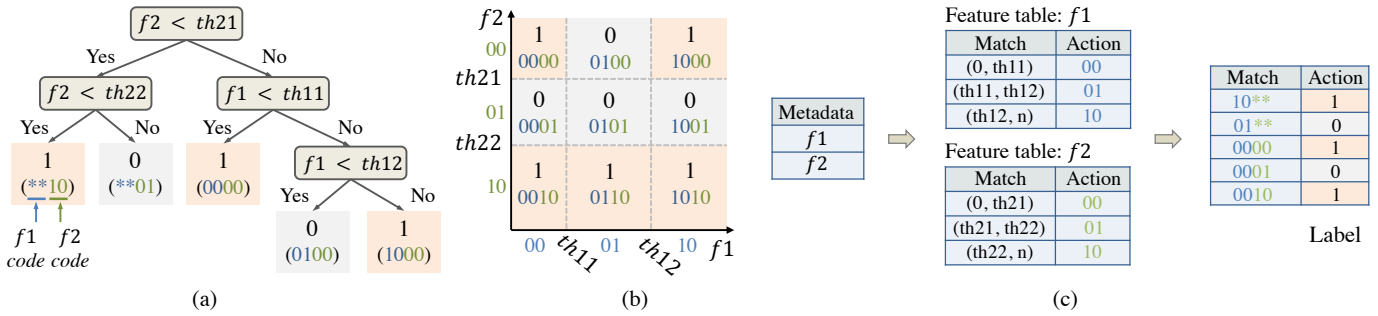



Fig. 7. Encode-based method proposed in [7] for a simple Decision Tree (DT) model mapping: (a) model structure of a trained DT, (b) feature space of a trained DT, (c) M/A tables for in-network inference.

encapsulating features in a digest structure and sending the digest. The former solution brings extra overhead by sending the whole packet and needs further processing on the control plane to parse features from the packet. In contrast, a digest-based solution saves the trouble by forwarding shorter messages, only carrying the features needed for analysis. The digest rate is lower than the packet rate, resulting in reduced logging overhead [23].

To implement digest-based logging, features $[f_1, f_2, \ldots f_n]$, stored in metadata, are packed in digests and marked with packet information (e.g., IP addresses). Digests sent to the control plane by calling *SendDigestEntry* function in *p4runtime_lib* [23]. The control plane is configured with P4Runtime to listen to digest messages. When it receives digests, encapsulated information is extracted and saved in *DataFrame* for unsupervised labeling and retraining.

### B. Unsupervised-based Labeling

Building upon proactive logging from the data plane, P4Pir automates the labeling process for the collected features received in digests (Figure 5 step 3).

It is important to label the records and update traffic profiles for in-network ML retraining, to ensure accurate inference results in the presence of emerging attacks that may cause distribution drift in incoming traffic. To achieve this without interrupting the runtime reconfiguration loop caused by manual labeling, P4Pir leverages an automated unsupervised labeling process in logs, employing outlier detection. By considering the emerging attack patterns as outliers compared to the benign pattern, an unsupervised learning algorithm is utilized to identify these outliers in logged records and effectively learn about emerging attacks at the gateway.

To select a suitable unsupervised-based algorithm, classic algorithms for anomaly detection have been studied: One-Class SVM, Local Outlier Factor (LOF), and Isolation Forest (iForest). *One-Class SVM* algorithm is designed to identify abnormal data by applying SVM to "one-class" problem, where a hyperplane is found to approximate positive examples, labeling data within this area as positive and data outside as negative [50]. *LOF* utilizes nearest neighbors to estimate local density and computes relative density to identify outliers [51]. *iForest*, on the other hand, builds tree structures to identify anomalies based on the assumption that abnormal samples can be isolated with fewer random feature splits and are closer to the tree's root than normal samples [52].

We assess each algorithm's labeling performance by comparing their learning results on datasets listed in Section IX-B. As an example, Figure 8 illustrates the accuracy of these algorithms in identifying UDP flooding attacks from *EDGE-IIOTSET* [45]. *AUC* score (Area under the ROC Curve) is used to depict the accuracy performance as the area under the TPR/FPR curve. The results show that both LOF and One-Class SVM exhibit high false alerts, incorrectly labeling benign traffic. In contrast, iForest achieves a balanced labeling capability for both benign and malicious traffic, with an AUC score of 80%. Given its better performance, *iForest* is selected for automated record labeling in P4Pir.

After this process, the newly logged and labeled records serve as the input for tree-based model (i.e. DT/RF) retraining and mapping to generate a new set of M/A table rules for in-network inference. Note that while the unsupervised learning algorithm (*iForest*) is utilized for automated labeling in the logs, the actual in-network mitigation is conducted by the supervised DT/RF model, highlighting the distinction between the two function blocks in P4Pir (as plotted in Figure 4).

## VIII. Model Remapping and Shadow Table Updates

Using the automatically collected logs, in-network inference model can be retrained and remapped to fresh M/A tables periodically. P4Pir facilitates shadow table updates at runtime, enabling the remapped in-network model to be configured hitlessly in the data plane without disrupting gateway functions. This section describes how shadow table updates work and how it achieves hitless in-network model updates (Figure 5 step 4).

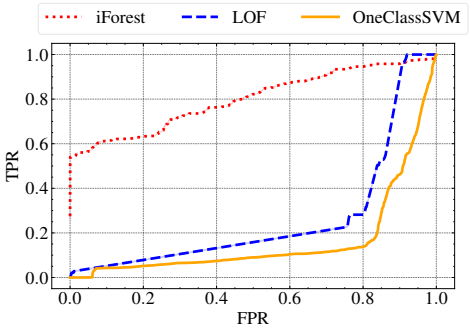



Fig. 8. Comparison of training accuracy of unsupervised-based labeling algorithms on UDP attacks in *EDGE-IIOTSET* [45]. $TPR = \frac{TP}{TP+FN}, FPR = \frac{FP}{TN+FP}$ indicate how each algorithm labels benign and malicious traffic. TPR (True Positive Rate) is calculated based on number of TP (True Positive) and FN (False Negative) samples, and FPR (False Positive Rate) is calculated based on number of FP (False Positive) and TN (True Negative) samples.

When a fresh set of M/A tables is generated by the control plane, these rules are then written to the data plane, forming new inference thresholds and completing the update process. This approach differs from the classical model retraining [5] in terms of update location, model objective, and update requirements. In the classical method, retraining and updating the model involves loading the trained model file onto processors like CPU. However, the in-network approach discussed here deviates from the classical approach due to the use of programmable data plane within IoT gateway. It uses a different inference process, which cannot be directly updated by loading the trained remodel file. Moreover, it is required that the update should not cause any stop or disruption to normal traffic processing.

**Challenge: seamless rule update.** Ensuring seamless runtime reconfiguration for the in-network model is challenging as modifications to M/A tables impact the data plane functionality. This challenge is first found in Software-Defined Networking (SDN) [53], which shares similar flow table update issues to prevent temporary disruption on packet routing. When it comes to in-network scenario on programmable data plane, atomicity limitations in P4Runtime Remote Procedure Call (RPC) operations further complicate the update process. Currently, only add/remove rule operations are supported [23], making it hard to directly overwrite parameter values in M/A rules. One approach is to remove all old rules and write new ones, but this risks inconsistent packet forwarding. To mitigate this impact, a method for seamless updates is desired.

**Solution: shadow table update.** P4Pir employs a shadow update method to minimize disruptions to data plane functions during updates (as in Figure 9). This method utilizes two tables: an active M/A table and a shadow M/A table. New rules are first updated in the shadow table, and then a flag is triggered to swap the roles of the active and shadow tables.

The following is a detailed workflow. When a new set of M/A table rules is generated by *Planter*'s remapping in the control plane (Figure 9 step 5 & 6), these rules are inserted via RPC to pre-configured shadow M/A tables in pipeline (Figure 9 Step A). To decide the timing of when shadow tables should take effect, an *UpdateFlag* is stored as a table entry. When this flag is 0, new rules in shadow tables remain deactivated, and incoming traffic are processed by rules in active tables. At runtime, *UpdateFlag* is set as 1, triggering a switching of active/shadow tables. This action activates shadow tables and drives ingress pipeline to apply new rules. Thereafter, incoming traffic is processed through new rules and obtains inference decisions of the updated model.

With shadow update design, in-network model updates are atomic, allowing seamless swapping of active and shadow tables without waiting for model-related rule operations. It eliminates disruption to forwarding functions and model inference process, albeit at the cost of increased resource utilization.

## IX. Experimental Evaluation

The evaluation aims to examine the following questions:

- Can P4Pir enable continuous learning and accurate detection of emerging attacks? (§IX-D)
- How do the number of features and logging frequency affect detection performance? (§IX-D)
- Can P4Pir quickly mitigate malicious traffic? (§IX-E)
- Can P4Pir achieve hitless model updates, without interrupting IoT gateway functionality? (§IX-F)
- Does P4Pir's in-network analysis function burden network throughput and CPU usage in an IoT gateway? (§IX-F)

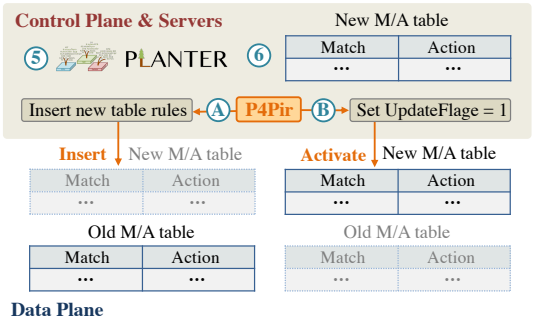



Fig. 9. Shadow rule update scheme in P4Pir.

### A. Experimental Setup

P4Pir prototype was developed on P4Pi [54], using Raspberry Pi (RPi) 4 Model B with 8GB of RAM, and running P4Pi release v0.0.3 using bmv2 with v1model architecture [55]. We also prototyped P4Pir on the Dell EMC Edge Gateway 5200 hardware device [56], where P4Pir's functionality is encapsulated within Docker and executed on the Dell gateway device. P4Pir's code is implemented mainly in P4 and Python to provide data plane (in Section V-VI) and control plane (in Section VI-VIII) functionality, correspondingly. ML Model training and mapping for in-network inference function (introduced in Section VI) is based on *scikit-learn* [48] and *Planter* [46].

For performance evaluation, P4Pir was connected to another RPi and a desktop with an Intel(R) Xeon(R) W-2133 CPU @ 3.60GHz and 64 GB RAM as client and server. Public IoT datasets *CIC-IDS2017* [57], *EDGE-IIOTSET* [45] and YTY2018 [4] are used for model training and evaluation. The captured traces in the dataset are replayed using *tcpreplay*. In these datasets, attacks are launched in different time slots using a week-worth of data. In order to examine the effectiveness of continuous learning in P4Pir, we assume an initial state in which the gateway learns only one attack on the first day. Then, another type of attack is replayed to simulate emerging attacks on the next day. After P4Pir is deployed, we compare its accuracy with a baseline's accuracy obtained using a static model. The static model is initialized by learning a single attack on the first day and maintained unchanged for incoming traffic replayed from other days.

### B. Datasets

Public datasets used to evaluate P4Pir's performance under different traffic scenarios are described as below.

**CIC-IDS2017** This dataset [57] collects 5-days traffic records for IDS analysis, containing various attacks like scanning, DoS/DDoS, etc. The attacks are launched separately in

each day's time slot. In this dataset, we assume that the model only learns a scanning attack on the first day as the initial state, and use DoS and Botnet attacks as the emerging ones.

**EDGE-IIOTSET** The dataset [45] includes benign traffic collected from sensors via diverse IoT protocols and IoT protocol-related attacks. To run a different attack scenario from *CIC-IDS2017*, in this dataset, we assume that the model learns only DDoS TCP SYN attack on the first day as the initial state, and uses other attacks (vulnerability scanning, UDP flooding, and HTTP flooding) as the emerging ones.

### C. Evaluation Metrics

**Detection accuracy:** There are several metrics to evaluate detection performance. In this paper, we use accuracy (ACC), F1 score, and true positive rate/false negative rate (TPR/FNR) as the metrics defined as below to evaluate detection performance, where $Precision = \frac{TP}{TP+FP}$ and $Recall = \frac{TP}{TP+FN}$. $TP$, $FP$, $TN$, and $FN$ are defined the same as in Figure 8.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

$$F_1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

$$TPR = \frac{TP}{TP + FN}, FNR = \frac{FN}{FN + TP} \tag{4}$$

**Throughput/jitter/drop rate:** Network throughput, jitter, and drop rate are measured using *iperf2*, with an RPi and a desktop used as the client and server, respectively. Every round of experiment generates TCP/UDP traffic and throughput/jitter/drop rate results are reported by *iperf2*.

**CPU resources:** CPU resource consumption is monitored to quantify P4Pir's load on the gateway. CPU utilization is recorded with the command `cat /proc/stat` to print the cycle usage breakdown in each CPU core. The tool `vcgencmd` is run to obtain core temperature.

### D. Detection Performance

**Detection efficiency.** Table III and Table IV list accuracy and F1 score results of encode-based Decision Tree (DT) and Random Forest (RF) [46] with and without P4Pir's model updates. Static DT/RF models without P4Pir's updates are initialized as a baseline and trained with a single attack ("Base" columns in Table III and Table IV). To keep the results comparable, 5-tuple features {*source IP, destination IP, source port, destination port, protocol*} are used to train static DT/RF model in both datasets. To select model parameters, we use grid search methods and use the parameters that give high accuracy. That is, in *CIC-IDS2017* [57], DT model is trained with a maximum depth of 5, and RF model is trained with 5 trees and maximum depth of 5. In *EDGE-IIOTSET* [45], DT model is trained with a maximum depth of 6, and RF model is trained with 6 trees and maximum depth of 6.

The results show: 1) DT/RF mapped to the data plane for in-network inference can achieve the same level of accuracy as the benchmark performance given by both datasets [45, 57], reaching more than 90% accuracy and F1 score as listed in initial states. 2) Different attack patterns may result in different levels of accuracy decrement for static models (Baseline), and P4Pir can efficiently mitigate it through model updates. Accuracy decrement can be seen for baselines in "SCAN→DOS" column in Table III and "SYN→SCAN" column in Table IV. It may be due to the changing attack attributes and feature distributions that static models have not learned. In P4Pir, the unsupervised labeling mechanism learns changing feature distributions in logs and retrains DT/RF with the logs to detect attacks with increased accuracy. It increases DT's accuracy by more than 50% and RF's accuracy by more than 30% ("SCAN→DOS" column in Table III and "SYN→SCAN" column in Table IV). 3) With different models initialized, P4Pir has different levels of performance improvement. P4Pir update method improves DT's performance ("SYN→SCAN" column in Table IV) given that DT is less scalable than RF.

TABLE III
DETECTION ACCURACY ON DATASET CICIDS 2017.

| | | SCAN | SCAN→DOS | | SCAN→BOT* | |
|---|---|---|---|---|---|---|
| | | Init | Base | P4Pir | Base | P4Pir |
| DT | ACC | 0.987 | 0.604 | 0.932 | 0.900 | 0.923 |
| | F1 | 0.984 | 0.568 | 0.868 | 0.776 | 0.820 |
| RF | ACC | 0.989 | 0.731 | 0.942 | 0.987 | 0.989 |
| | F1 | 0.985 | 0.027 | 0.869 | 0.964 | 0.987 |

TABLE IV
DETECTION ACCURACY ON DATASET EDGE-IIOTSET.

| | | SYN | SYN→SCAN | | SYN→UDP | | SYN→HTTP† | |
|---|---|---|---|---|---|---|---|---|
| | | Init | Base | P4Pir | Base | P4Pir | Base | P4Pir |
| DT | ACC | 0.910 | 0.156 | 0.945 | 0.435 | 0.903 | 0.921 | 0.941 |
| | F1 | 0.953 | 0.270 | 0.972 | 0.606 | 0.949 | 0.924 | 0.970 |
| RF | ACC | 0.999 | 0.674 | 0.999 | 0.888 | 0.903 | 0.791 | 0.902 |
| | F1 | 0.999 | 0.788 | 0.999 | 0.934 | 0.944 | 0.876 | 0.943 |

* Init - Initial state, Base - Baseline from in-network inference model in [46], SCAN - port scanning attack, DoS - DDoS LOIT attack, BOT - Botnet ARES attack. "→" indicates the change in attack pattern from the initial state to an emerging attack.
† Init - Initial state, Base - Baseline, SYN - DDoS TCP SYN attack, SCAN - vulnerability scanning attack, HTTP - HTTP flooding attack, UDP - UDP flooding attack.

**Number of features impact on accuracy.** *EDGE-IIOTSET* dataset is used and features are selected from the ones with high importance as listed in Table II. Figure 10 presents how different numbers of features affect the detection's accuracy after an update. For both DT and RF models, using more features for learning can increase the accuracy and lower the False Positive Rate (FPR). When 15 features are used, both models can reach an accuracy higher than 95%. The improvement in FPR performance is more significant, indicating that more benign packets can be correctly labeled when more features are used to reflect the traffic pattern.

**Update interval's impact on accuracy.** Figure 11 presents how different update intervals affect the detection accuracy based on *EDGE-IIOTSET* dataset. RF is used as an example. In general, longer update intervals provide higher accuracy and F1 score. This is because more traffic is logged by the data plane and the control plane can learn from more logged traffic to generate the updated model and parameters. Despite the improvement, a longer update time does not necessarily reflect better results. It can reach a fairly good detection

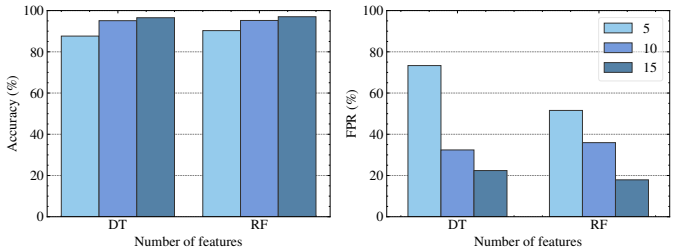



Fig. 10. Accuracy/FPR vs. number of features (5/10/15) when P4Pir is deployed. DT/RF initialized with SYN attack and learning the HTTP attack.

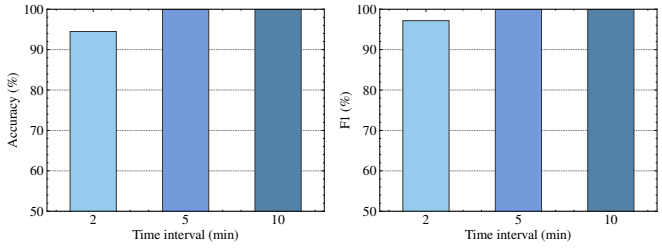



Fig. 11. Accuracy/F1 vs. time interval. RF is initialized with SYN attack and updated with different time intervals in P4Pir to learn the HTTP attack.

(∼99%) when the update is triggered every 5 minutes and the performance would be similar to updates every 10 minutes.

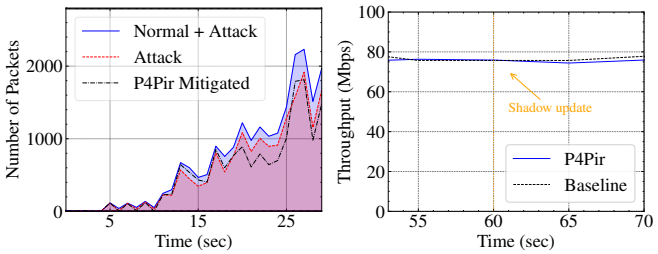
### *E. Mitigation Performance*

To assess the effectiveness of our in-network design in fast attack mitigation, Figure 12 (a) presents a snapshot of captured traffic. In this capture, Friday afternoon record with normal and DDoS traffic in *CIC-IDS2017* dataset is replayed. Total amount of traffic (including normal and DDoS attacks) is marked in blue, and attack traffic is marked in red. DT is deployed on P4Pir based on five features and the black dash line depicts the number of packets that are mitigated (dropped) by P4Pir. P4Pir learns the new attack in sub-millisecond and quickly starts dropping attack traffic, as shown by the overlap of red and black dash lines in Figure 12 (a).

When the attack starts in the third second, traffic volume increases as the red line climbs up. P4pir can mitigate the malicious traffic immediately as the black line increases together with the red line. When the attack reaches a peak at 16 seconds, P4Pir can efficiently mitigate the malicious traffic (indicated by the overlapped red line and black line). Average mitigation accuracy aligns with the accuracy results presented in Table III, which is around 93%. In some cases, FPR detection might lead to a false packet drop, indicated by the black line exceeding the red line. Such false alerts can be reduced by introducing more features (e.g., application-layer features from MQTT) as presented in Figure 10. To summarize, in-network analysis function applied by P4Pir can mitigate attacks promptly and prevent the traffic explosion impact of malicious attacks.

### *F. System Performance*

**Update efficiency.** P4Pir introduces a shadow update scheme (Section VIII). As the update involves changes to table rules within the data plane, the efficiency of shadow updates




(a) Mitigation performance. (b) Shadow update impact.

Fig. 12. (a) A snapshot of mitigation performance of P4Pir-DT on DDoS attack from *CIC-IDS2017* dataset. (b) Shadow update impact of P4Pir-DT on throughput. Baseline - forwarding setup without any model update.

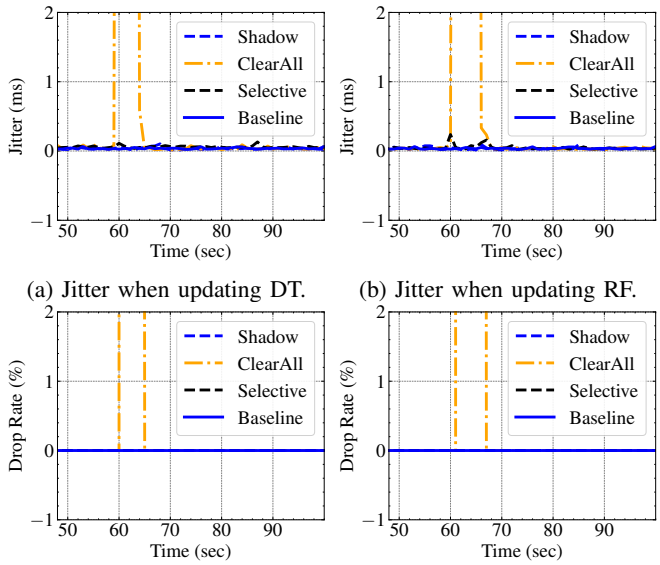
is evaluated for two aspects: 1) whether the update interrupts other data plane functions, and 2) whether the update affects the detection accuracy.

The first aspect involves assessing network metrics such as throughput, jitter, and drop rate. A qualitative comparison of throughput is presented in Figure 12 (b). At the 60-second mark (indicated by the yellow arrow), the shadow rule update does not affect throughput, as indicated by the similarity between the blue solid curve (shadow update) and the black dashed curve (baseline without P4Pir deployment). To investigate potential interruptions and impacts on the forwarding queue, jitter, and drop rate, different update schemes are compared in Figure 13 (a) and (b), including ClearAll (flushing and inserting all table entries) and Selective update schemes. Both ClearAll and Selective schemes demonstrate a negative effect on jitter performance, as they directly manipulate M/A rules on the data plane at runtime, which takes time and affects data plane forwarding. In contrast, shadow update in P4Pir is better in ensuring updated rules are in place immediately with negligible impact. Drop rate stays the same during shadow update, but it is high for ∼5sec when using ClearAll.

The second aspect is evaluated by the detection accuracy metrics in terms of false negative rate (FNR) and true positive rate (TPR). Results are plotted in Figure 14. Flushing and inserting table rules for model updates can disrupt the decision accuracy, resulting in the failure to identify malicious attacks and slow improvement of the FNR. By comparison, shadow update provides a more immediate improvement of FNR and TPR (∼2sec faster). TPR/FNR was 0.45/0.55 before the update but improved from second 60 after triggering the model update. Selective or ClearAll updates bring a slow improvement of FNR/TPR after the update.

Figure 13 and 14 demonstrate that shadow update scheme in P4Pir outperforms other update schemes in achieving seconds faster model updates, improving detection accuracy in second with a negligible impact on forwarding function.

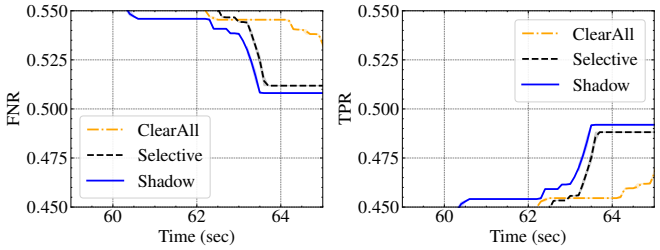
**Impact on network throughput/jitter.** Figure 15 (a) presents how different in-network ML deployments on P4Pir impact throughput. In the baseline scenario, measuring only basic forwarding functionality, throughput reaches approximately 92Mbps. When P4Pir is deployed, throughput decreases as a function of number of features parsed. Such a throughput decrement is because of the limited processing capability of bmv2 running on RPi. Compared with RF, DT is less sensitive to the number of features in terms of throughput




(a) Jitter when updating DT. (b) Jitter when updating RF.

(c) Drop rate when updating DT. (d) Drop rate when updating RF.

Fig. 13. Jitter and Drop rate when DT/RF is updated using different update schemes. A model update takes place at 60 seconds. Shadow - shadow update scheme in P4Pir. Existing solutions: ClearAll - an update scheme reinstalling all rules, Selective - an update scheme reinstalling model-related rules. Baseline - forwarding setup without any analysis deployment.

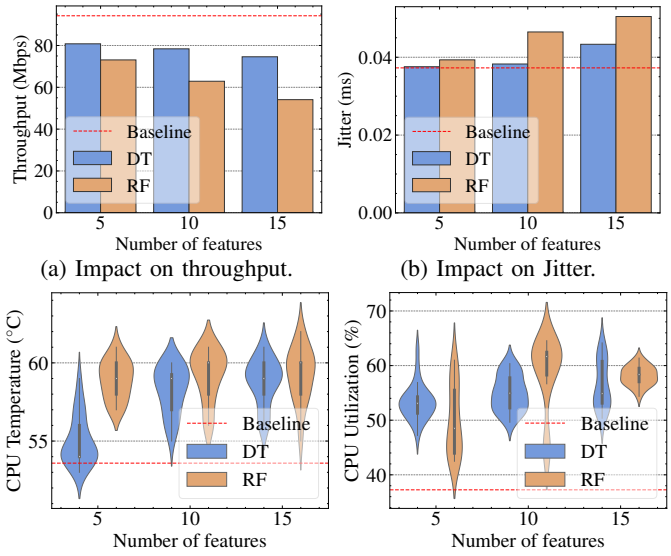



(a) False negative rate (FNR). (b) True positive rate (TPR).

Fig. 14. (a) FNR curve and (b) TPR curve when DT is updated with ClearAll/Selective/Shadow scheme.

decrement. That is because the model structure in RF requires more feature inference than DT. The number of parsed features and type of model brings similar impact on traffic jitter as depicted in Figure 15 (b). Compared to throughput, the increase in jitter due to complex traffic processing is minimal (with <0.05ms increment). Combined with the accuracy results in Table IV, a trade-off between model accuracy and overhead can be observed. When DT is trained with 5 features, P4Pir's update function can significantly enhance DT's detection capability by more than 50% and only cause 14% throughput reduction and negligible jitter increase.

**Impact on CPU resources.** In Figure 15 (c) and (d), we can see how much CPU resources P4Pir consumes in terms of CPU temperature and CPU utilization. It indicates how much processing load is added by P4Pir. Parsing more in-band features generally requires more resources, leading to higher CPU utilization and CPU temperature (∼10% increment in CPU temperature and ∼21% increment in CPU utilization compared to the baseline). Complex models like RF present slightly higher processing load than DT. This aligns with the observation in Figure 15 (a) and (b), where the deployment of DT with P4Pir update achieves the optimal balance of high detection accuracy and low overhead.




(a) Impact on throughput. (b) Impact on Jitter.

(c) Impact on CPU temperature. (d) Impact on CPU utilization.

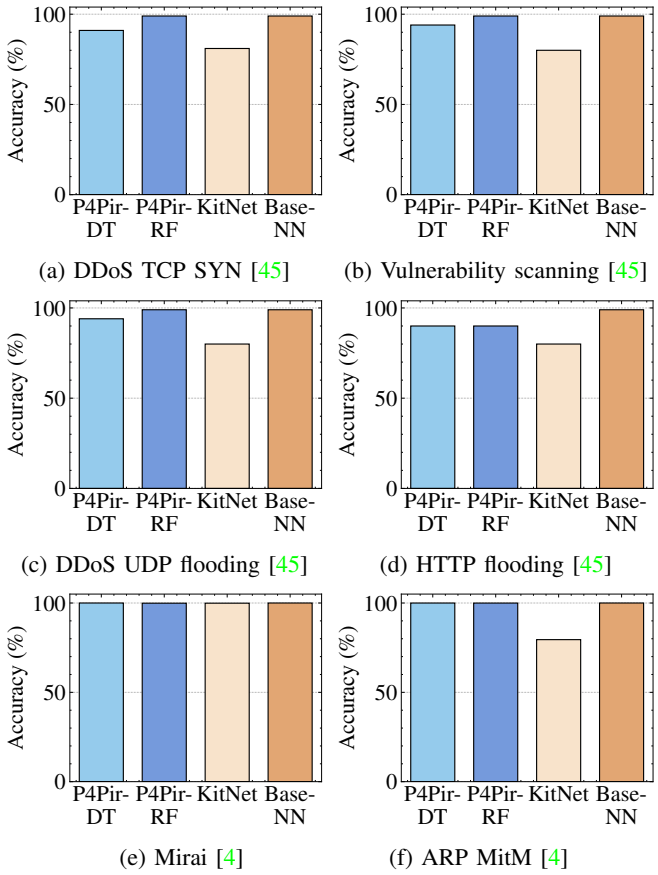
Fig. 15. P4Pir's impact on network throughput, jitter and CPU resource consumption. Baseline presents the setup when basic forwarding is configured without any analysis function.

## X. Comparison with State-of-the-Art

For IoT attack detection, we employ two state-of-the-art frameworks: KitNet (core detection model of Kitsune [4]) and Passban IDS [30]. To ensure a consistent hardware environment, we deploy their open-source solutions on Raspberry Pi, similar to our solution. However, since these frameworks do not offer mitigation capabilities, we compare model detection performance using two public datasets: *EDGE-IIOTSET* [45] (used in Table IV) and *YTY2018* [4] (used in KitNet/Kitsune). To make a fair comparison, we use attacks with different patterns, such as Mirai and ARP MitM. Additionally, we establish a baseline reference on a server for both datasets by running a Neural Network (NN) model (3 layers with 48 neurons/layer based on grid search accuracy in TensorFlow [58]).

Figure 16 (a)-(d) present an accuracy comparison between P4Pir and online-learning model KitNet proposed in Kitsune [4]. A reference baseline with NN model reaches ∼99% accuracy on a server. Results show that when DT is deployed with P4Pir, the results are higher than KitNet, reaching around 90% accuracy. When RF is deployed with P4Pir, its accuracy can reach a similar level as the baseline NN model (∼99%) in vulnerability scanning and TCP SYN/UDP flooding attacks. KitNet achieves ∼80% accuracy on these attacks. We observed performance variations when using KitNet on a different dataset [45], rather than the original one used in [4]. Dataset [45] employed here consists of packet-level features, while original dataset [4] extracted traffic incremental statistics. Performance variations can be attributed to features used to reflect traffic patterns, which renders KitNet less scalable.

The comparison with Kitsune's KitNet is also done in Kitsune's dataset [4] as Figure 16 (e) and (f). Mirai and ARP MitM attacks are used in this test case and results present that P4Pir can achieve a similar accurate performance as KitNet on Mirai attack and 10% more accurate performance on ARP MitM attack. P4Pir can also achieve comparable performance as baseline NN running at the server. The degradation of Kit-




(a) DDoS TCP SYN [45] (b) Vulnerability scanning [45]

(c) DDoS UDP flooding [45] (d) HTTP flooding [45]

(e) Mirai [4] (f) ARP MitM [4]

Fig. 16. P4Pir vs. related works in terms of accuracy and throughput with different attacks under *EDGE-IIOTSET* [45] and *Kitsune* [4] traces. P4Pir-DT/P4Pir-RF means DT/RF deployed with P4Pir. KitNet is the algorithm used in *Kitsune* [4]. Base-NN is Neural Network run at server as a baseline.

Net's performance in ARP attack is explained as its vulnerable to out-of-distribution (o.o.d.) samples [59].

Passban IDS proposed in [30] achieves traffic analysis and attack detection based on a supervised learning-based method for IoT gateways. With in-network ML design and model update in P4Pir, the detection accuracy can reach similar level as Passban IDS (F1>80% for scanning and flooding attack detection). In terms of system performance, P4Pir has a more lightweight performance than Passban IDS, beneficial from offloading ML inference to the data plane. a) Throughput is compared where P4Pir achieves higher network throughput than Passban IDS (∼14% decrease vs. ∼17% decrease, compared with Baseline forwarding). 2) Passban IDS has high CPU usage when the system runs in high throughput (>70% when throughput is higher than 70Mbps), while P4Pir has ∼60% CPU usage at that throughput.

## XI. Discussion

Evaluation results demonstrate that P4Pir can efficiently mitigate attacks within IoT gateway via continuous learning and ML updates. Observations and trade-offs are discussed.

**Model selection** In this work, we present DT and RF as tree-based ML inference models for efficient in-network analysis, while acknowledging that other models (e.g. XGB, SVM, NN) can also be deployed on programmable data plane for in-network attack detection [46]. However, there is a trade-off between model accuracy and overheads. Model with complex computations may achieve good performance, but also bring extra overheads to the data plane. IoT gateway deployment calls for models with low overheads and high accuracy. Our experimental results show that the in-network deployment of DT has lower overhead but is less effective in detecting emerging attacks compared to RF.

**Model updates** P4Pir offers two update options: parameter updates and feature updates. As the distribution of incoming traffic changes, the model can be directly updated by inserting new parameters and thresholds. In situations where accuracy drastically drops, it indicates that the current set of features may not adequately represent the current traffic pattern. In such cases, new features are required, which may necessitate program re-initialization involving a new feature extraction and in-network model mapping process.

**Use cases** P4Pir is designed for attack mitigation, serving as a sample use case. Likewise, it can also be used for other ML-based applications such as IoT device identification [60], failure mitigation, or traffic scheduling. P4Pir's in-network design enables them to achieve prompt ML-based analysis and runtime reconfiguration in dynamic traffic scenarios.

## XII. Conclusion

We introduced P4Pir, an in-network ML-based analysis solution for IoT gateways, implemented on low-cost P4Pi platform using Raspberry Pi and Dell Edge Gateway. P4Pir leverages the programmable data plane to enable fast attack mitigation and seamless ML updates for continuous learning against emerging threats. With P4 language, we achieve flexible in-band feature collection and in-network ML inference, ensuring swift attack detection and mitigation. To keep the model updated in identifying traffic patterns within the IoT gateway, P4Pir actively logs in-band extracted features from the data plane to the control plane. These features are then labeled using an unsupervised *iForest* algorithm. These records facilitate model retraining, which is then seamlessly mapped to the data plane for in-network inference through shadow table updates. Evaluation results show that P4Pir has minimal impact on network performance. P4Pir highlights the advantages of deploying in-network ML inference in IoT gateways. We hope it can inspire future explorations in topics like federated deployment for scenarios with multiple gateways.

## Acknowledgement

We thank Radostin Stoyanov for his contribution to early work on this project, bringing up and optimizing the P4Pi platform. We also thank Damu Ding and Eder Ollora Zaballa for the discussion and their constructive comments. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

## References

[1] I. Parvez, A. Rahmati, I. Guvenc, A. I. Sarwat, and H. Dai, "A survey on low latency towards 5g: Ran, core network and caching solutions," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 3098–3130, 2018.

[2] "Service requirements for the 5G system (3GPP TS 22.261 version 16.14.0 Release 16)," 3GPP, Standard, Apr. 2021.

[3] A. G. Alcoz, M. Strohmeier, V. Lenders, and L. Vanbever, "Aggregate-based congestion control for pulse-wave ddos defense," in *Proceedings of the ACM SIGCOMM 2022 Conference*, New York, NY, USA, 2022, p. 693–706.

[4] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *arXiv preprint arXiv:1802.09089*, 2018.

[5] R. Kolcun *et al.*, "The case for retraining of ML models for iot device identification at the edge," *arXiv preprint arXiv:2011.08605*, 2020.

[6] J. H. Lee and K. Singh, "SwitchTree: in-network computing and traffic analyses with Random Forests," *Neural Computing and Applications*, 2020.

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

[8] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 87–95, jul 2014.

[9] A. Pashamokhtari, N. Okui, Y. Miyake, M. Nakahara, and H. H. Gharakheili, "Inferring connected iot devices from ipfix records in residential isp networks," in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, 2021, pp. 57–64.

[10] S. Klein, *Ingesting Data with Azure IoT Hub*. Berkeley, CA: Apress, 2017, pp. 57–70.

[11] K. Palani, E. Holt, and S. Smith, "Invisible and forgotten: Zero-day blooms in the iot," in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops*, 2016, pp. 1–6.

[12] C.-C. Teng, J.-W. Gong, Y.-S. Wang, C.-P. Chuang, and M.-C. Chen, "Firmware over the air for home cybersecurity in the internet of things," in *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2017, pp. 123–128.

[13] Dell Technologies, "Dell emc edge gateway 5200 software user's guide," 2022. [Online]. Available: https://www.dell.com/support/manuals/en-ae/dell-edge-gateway-5200/egw-5200-software-users-guide

[14] G. C. Hillar, *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd, 2017.

[15] M. Hemmatpour, M. Ghazivakili, B. Montrucchio, and M. Rebaudengo, "Diig: A distributed industrial iot gateway," in *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 1, 2017, pp. 755–759.

[16] S. Srinivasa, J. M. Pedersen, and E. Vasilomanolakis, "Open for hire: Attack trends and misconfiguration pitfalls of iot devices," in *Proceedings of the 21st ACM Internet Measurement Conference*, New York, NY, USA, 2021, p. 195–215.

[17] M. Lima, R. Lima, F. Lins, and M. Bonfim, "Beholder – a cep-based intrusion detection and prevention systems for iot environments," *Computers & Security*, vol. 120, p. 102824, 2022.

[18] Y. Kim, J. Nam, T. Park, S. Scott-Hayward, and S. Shin, "SODA: A software-defined security framework for IoT environments," *Computer Networks*, vol. 163, p. 106889, 2019.

[19] P. Kumar, G. P. Gupta, and R. Tripathi, "An ensemble learning and fog-cloud architecture-driven cyber-attack detection framework for iomt networks," *Computer Communications*, vol. 166, pp. 110–124, 2021.

[20] P. K. Sharma, S. Singh, and J. H. Park, "Opcloudsec: Open cloud software defined wireless network security for the internet of things," *Computer Communications*, vol. 122, pp. 1–8, 2018.

[21] A. Mahmood *et al.*, "Industrial iot in 5g-and-beyond networks: Vision, architecture, and design trends," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 6, pp. 4122–4137, 2022.

[22] L. L. Peterson, C. Cascone, B. O'Connor, T. Vachuska, and B. Davie, *Software-defined networks: A systems approach*. Systems Approach LLC, 2021.

[23] The P4.org API Working Group, "P4runtime specification," 2021. [Online]. Available: https://p4.org/p4-spec/p4runtime/main/P4Runtime-Spec.pdf

[24] C. Zheng *et al.*, "IIsy: Practical In-Network Classification," *arXiv preprint arXiv:2205.08243*, 2022.

[25] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pForest: In-network inference with random forests," *arXiv preprint arXiv:1909.05680*, 2019.

[26] X. Zhang, L. Cui, F. P. Tso, and W. Jia, "pheavy: Predicting heavy flows in the programmable data plane," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4353–4364, 2021.

[27] G. Xie *et al.*, "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *IEEE INFOCOM 2022*, 2022, pp. 1938–1947.

[28] X. Hong, C. Zheng, S. Zohren, and N. Zilberman, "Linnet: Limit Order Books within Switches," in *Proceedings of the SIGCOMM '22 Poster and Demo Sessions*, New York, NY, USA, 2022, p. 37–39.

[29] ——, "Lobin: In-network machine learning for limit order books," in *2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR)*, 2023, pp. 159–166.

[30] M. Eskandari, Z. H. Janjua, M. Vecchio, and F. Antonelli, "Passban IDS: An Intelligent Anomaly-Based Intrusion Detection System for IoT Edge Devices," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6882–6897, 2020.

[31] Q. Qin, K. Poularakis, and L. Tassiulas, "A learning approach with programmable data plane towards iot security," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 410–420.

[32] F. Musumeci, V. Ionata, F. Paolucci, F. Cugini, and M. Tornatore, "Machine-learning-assisted ddos attack detection with p4 language," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[33] Y. Jia, F. Zhong, A. Alrawais, B. Gong, and X. Cheng, "Flowguard: An intelligent edge defense mechanism against iot ddos attacks," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9552–9562, 2020.

[34] I. Hafeez, M. Antikainen, A. Y. Ding, and S. Tarkoma, "Iot-keeper: Detecting malicious iot network activity using online traffic analysis at the edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 45–59, 2020.

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

[36] D. Ding, M. Savi, F. Pederzolli, M. Campanella, and D. Siracusa, "In-network volumetric ddos victim identification using programmable commodity switches," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1191–1202, 2021.

[37] A. Atutxa, D. Franco, J. Sasiain, J. Astorga, and E. Jacob, "Achieving low latency communications in smart industrial networks with programmable data planes," *Sensors*, vol. 21, no. 15, 2021.

[38] M. Zang, E. O. Zaballa, and L. Dittmann, "SDN-based In-Band DDoS Detection Using Ensemble Learning Algorithm on IoT Edge," in *2022 25th Conference on Innovation in Clouds, Internet and Networks*, 2022, pp. 111–115.

[39] H. Hindy, C. Tachtatzis, R. Atkinson, E. Bayne, and X. Bellekens, "Developing a siamese network for intrusion detection systems," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, ser. EuroMLSys '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 120–126.

[40] Q. Wang *et al.*, "MPInspector: A systematic and automatic approach for evaluating the security of IoT messaging protocols," in *30th USENIX Security Symposium (USENIX Security 21)*, Aug. 2021, pp. 4205–4222.

[41] T. Yu, V. Sekar, S. Seshan, Y. Agarwal, and C. Xu, "Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the internet-of-things," in *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XIV, New York, NY, USA, 2015.

[42] J. Kazemitabar, A. Amini, A. Bloniarz, and A. S. Talwalkar, "Variable importance using decision trees," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[43] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.

[44] Scikit-learn, "Permutation feature importance," 2022. [Online]. Available: https://scikit-learn.org/stable/modules/permutation_importance.html

[45] M. A. Ferrag, O. Friha, D. Hamouda, L. Maglaras, and H. Janicke, "Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications: Centralized and federated learning." IEEE Dataport, 2022.

[46] C. Zheng *et al.*, "Automating In-Network Machine Learning," *arXiv preprint arXiv:2205.08824*, 2022.

[47] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" in *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[48] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[49] B. M. Xavier, R. S. Guimarães, G. Comarela, and M. Martinello, "Programmable switches for in-networking classification," in *IEEE INFOCOM 2021*, 2021, pp. 1–10.

[50] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, "Support vector method for novelty detection," in *Advances in Neural*

*Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999.

[51] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2000, p. 93–104.

[52] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[53] J. H. Han *et al.*, "Blueswitch: Enabling provably consistent configuration of network switches," in *2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems*. IEEE, 2015, pp. 17–27.

[54] S. Laki, R. Stoyanov, D. Kis, R. Soulé, P. Vörös, and N. Zilberman, "P4Pi: P4 on Raspberry Pi for networking education," *SIGCOMM Comput. Commun. Rev.*, vol. 51, no. 3, p. 17–21, jul 2021.

[55] R. Stoyanov, A. Wolnikowski, R. Soulé, S. Laki, and N. Zilberman, "Building an internet router with P4Pi," in *Proceedings of the Symposium on Architectures for Networking and Communications Systems*, 2022, p. 151–156.

[56] DELL Technologies, "Dell EMC Edge Gateway 5200 software user's guide," 2023. [Online]. Available: https://dl.dell.com/content/manual77941191-dell-emc-edge-gateway-5200-software-user-s-guide

[57] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP*, 2018, pp. 108–116.

[58] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[59] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, "Ai/ml for network security: The emperor has no clothes," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2022, p. 1537–1551.

[60] 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*, New York, NY, USA, 2019, p. 25–33.