This is the author's accepted version of the article. The final version published by IEEE is A.T-J. Akem and M. Fiore, "Towards Real-Time Intrusion Detection in P4-Programmable 5G User Plane Functions," 2024 IEEE 32nd International Conference on Network Protocols (ICNP), 2024, pp. TBD, doi: TBD.

Towards Real-Time Intrusion Detection in P4-Programmable 5G User Plane Functions

Aristide Tanyi-Jong Akem*† and Marco Fiore*
*IMDEA Networks Institute, Spain, †Universidad Carlos III de Madrid, Spain
{aristide.akem, marco.fiore}@imdea.org

***Abstract*—Recent works have shown that Machine Learning (ML) models can be deployed in P4-programmable user planes for line rate inference on live traffic and that these user planes can also be used to accelerate the 5G User Plane Function (UPF). This work builds on these capabilities to explore how ML inference in the user plane can facilitate real-time intrusion detection in 5G networks. As a proof-of-concept, we describe how an ML model could be deployed into the UPF as a special Packet Detection Rule (PDR). We then train and deploy a tree-based classifier into a P4-programmable switch acting as the UPF and conduct experiments on a testbed with off-the-shelf hardware using experimental data from a 5G test network on a university campus. Our results confirm that running ML-based intrusion detection on P4-based UPFs ensures line-rate attack detection and classification with an accuracy of up to 98% in terms of F1 score, while keeping switch resource consumption increase under control.**

***Index Terms*—Machine learning, 5G, user plane function, P4**

I. Introduction

The rollout of 5G networks in recent years has led to an exponential growth in the number of connected devices. The surge in user numbers has increased malicious activity on the network, with attackers leveraging Internet of Things (IoT) devices to launch cyberattacks on the network [1]. In addition, 5G employs Software-Defined Networking (SDN) and Network Function Virtualization (NFV) which heavily rely on HTTP and REST API protocols, further exposing it to security vulnerabilities by providing access paths to attackers [2]. As such, efficiently and rapidly detecting cyberattacks on 5G networks is crucial to ensuring network security

In the SDN context, Machine Learning (ML) models running in the control plane have been proposed for network intrusion detection [3]–[5]. These models typically require back-and-forth communication with the user plane to collect traffic features or to inject rules resulting from model predictions. Such inter-plane communication traverses the slow path of the control and user plane separation (CUPS), inducing millisecond-level delays [6], which are undesirable in emerging applications that 5G is expected to support such as remote and robotic surgery or augmented and virtual reality.

Recent advances in user-plane programming have given rise to off-the-shelf programmable user-plane chips like Intel Tofino ASICs [7] and NVIDIA BlueField DPUs [8], as well as domain-specific programming languages like P4 [9]. This has sparked a strong interest in offloading network functions from the control plane to the user plane [10]. Such offloads enable programs to run at line rate, drastically reducing latency to sub-microsecond levels. To minimize delays in intrusion detection and improve response time, control-plane ML models are also

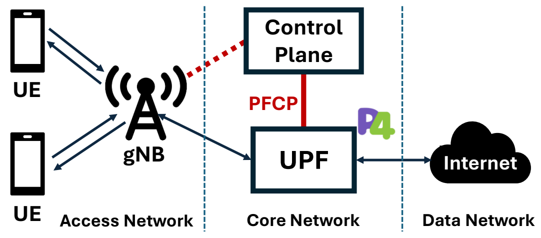

Fig. 1: 5G architecture overview with a P4-based UPF

being offloaded to the user plane, where they are embedded into programmable devices like switches for real-time inference on network traffic with high throughput and ultra-low latency [11]. Yet, switches are highly constrained in terms of available memory, supported mathematical operations, and the number of allowed operations per packet [12]. In this context, tree-based models like Decision Trees (DT) and Random Forests (RF) have emerged as the most scalable options for in-switch deployment due to a simple logical structure that makes them easier to map to the switch pipeline [13]–[17]. However, no prior work has tackled user-plane ML inference in the context of the 5G core network.

Users on 5G networks generate large traffic volumes which must be forwarded at very high speeds to ensure a high Quality of Service (QoS) and maintain sub-millisecond latency [18]. The User Plane Function (UPF) plays the essential role of forwarding this traffic between users (UEs) and external data networks, *e.g.*, the Internet, as shown in Figure 1. The control plane communicates with the UPF via the N4 interface using the Packet Forwarding Control Protocol (PFCP) [19]. Through this protocol, the Session Management Function (SMF) in the control plane creates sessions on the UPF to manage GPRS Tunneling Protocol (GTP) tunnels that enable users to communicate with external data networks and to set packet detection rules, forwarding action rules, buffering action rules, QoS enforcement rules, and usage reporting rules for users [20].

State-of-the-art UPFs are designed as multi-core software packet processors that run on off-the-shelf servers, leveraging kernel bypass tools like the Data Plane Development Kit (DPDK) for high performance [21]. With growing network traffic and user numbers, the UPF must perform its role efficiently and at increasingly higher speeds. To achieve this, recent works have demonstrated the possibility of building efficient and high-speed UPFs on programmable switches [21]–[23]. Such hardware-accelerated UPFs can potentially boost performance over traditional ones by about 31% per unit cost, and about 92% per unit of power [21]. Despite this high

©2024 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

performance, none of the existing proposals has explored the prospect of porting ML inference to the P4-based UPF to enhance its abilities and enable new functions like real-time and high-speed intrusion detection and traffic classification.

In this paper, we exploit recent advances in ML inference in P4-programmable switches and hardware-accelerated UPFs to propose an *ML-enabled UPF* which can perform line-rate and on-the-fly cyberattack detection in the 5G user plane. Our proposal involves deploying a trained model fully into the switch, where it infers on data packets without any communication with the control plane, hence minimizing inference latency. The main contributions of the paper are as follows.

- We propose embedding trained ML models into P4-based UPFs to enable high-speed inference on live 5G traffic in the user plane with ultra-low latency. As a proof-of-concept, we embed a tree-based classifier into a P4-programmable switch for line-rate intrusion detection and attack classification in 5G user planes.
- We implement the proposed model into an off-the-shelf Intel Tofino switch to verify its feasibility in hardware, and we make our source code publicly available[1].
- We conduct an experimental evaluation with real-world data from a test 5G network on a university campus, on which an intrusion detection and attack classification use case is designed. Experimental results show how our approach can detect and classify malicious traffic with over 98% accuracy and sub-μs latency.

We believe these contributions lay the groundwork for an ML-enabled 5G UPF whose decision-making capabilities are enhanced by embedded ML models, transcending current state-of-the-art rule-based approaches.

II. The User Plane Function (UPF)

The UPF is located in the 5G core network and connects the radio access network (RAN) via gNBs to data networks. It is the main packet processing engine of the 5G user plane and its workflow is defined by the 3GPP TS 29.244 [19]. The UPF processes packets for 5G users (UEs) according to the workflow in Figure 2, based on instructions from the control plane, specifically from the Session Management Function (SMF). The SMF controls the UPF's handling of packets by establishing, modifying or deleting PFCP sessions, and by adding, modifying or deleting rules, or activating/deactivating existing ones [19]. The rules used by the UPF to process packets are shown on the right of Figure 2 and are as follows.

Packet Detection Rules (PDRs). They enable the UPF to assign each packet to a specific UE and a traffic class [22]. The PDR specifies packet fields, *e.g.*, 5-tuples or tunnel headers which if fully matched identify the corresponding UE and traffic class which could be related *e.g.*, to QoS or pricing.

Forwarding Action Rules (FARs). FARs tell the UPF how to forward the matching packet. FARs are invoked by the matched PDR as shown in Figure 2, to ensure that traffic is correctly forwarded, even when the UE changes location.

[1]Source code: https://github.com/nds-group/5G_UPF_Intrusion_Detection.

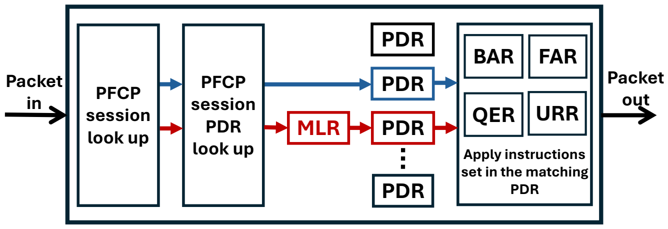

Fig. 2: Packet processing flow in the UPF

Buffering Action Rules (BARs). They indicate how downlink packets destined for a specific UE can be buffered in the UPF if the device is idle. BARs specify, *e.g.*, how large buffered packets can be or for long they can be buffered.

QoS Enforcement Rules (QERs). Each QER tags the packet to a given QoS class. The UPF then uses these rules to guarantee available bandwidth to each UE according to their QoS class. These rules enable the UPF to enforce traffic policing through per-user bandwidth caps [22].

Usage Reporting Rules (URRs). They specify when and how uplink and downlink network usage statistics for each UE should be reported to the control plane. These are essential to the operator for billing and usage control.

The ordinary packet processing flow in the UPF is depicted by the blue arrows in Figure 2. When a packet arrives at the UPF, it first performs a lookup of the provisioned PDRs to identify the packet's corresponding PFCP session. Once a session is found, it performs another lookup to find the first PDR that matches the packet among the possible PDRs of the PFCP session [19]. The UPF ensures that only the PDR with the highest precedence is selected. If no matching PDR is found, the packet is dropped. When a matching PDR is found, sets of instructions *e.g.*, BARs, FARs, QERs and/or URRs are applied as shown in Figure 2 to further process the packet.

The UPF must perform its role efficiently to support high throughputs of up to 1 Gbps/user while maintaining latency below 1 ms. These requirements constitute a challenge for most state-of-the-art UPFs which are designed as software packet processors, necessitating hardware acceleration [21]. Recent works have demonstrated the possibility of implementing performant P4-based UPFs in programmable network devices [22], [23], including a full offload of the PFCP processing [21]. This means that the full packet processing flow depicted in Figure 2 can be implemented in programmable switches. This creates room for deploying more functions *e.g.*, ML inference for intrusion detection, within the UPF by leveraging the possibilities that P4-programmable devices provide.

III. Intrusion detection in the P4-based UPF

We envisage building on existing P4-based UPF proposals such as MacDavid *et al.* [22], CeUPF [23], and AccelUPF [21] to enable line-rate ML inference with high throughput and ultra-low latency in the 5G UPF.

A. Introducing ML into the UPF workflow

As described in Section II, PDRs use specific matching fields from packets to determine what user is associated with the packet and to assign it to a traffic category. This is similar to what an ML-based traffic classifier would do, *i.e.*, extract and match a given set of features and then assign a traffic class to the packet. P4-based UPFs are implemented as a series of match and action tables and actions that process packets. Recent works have also shown how trained ML models can be deployed in programmable switches as a series of match and action tables for line-rate inference [14], [16], [24], [25].

To enable ML inference for intrusion detection in the P4-based UPF, we envision deploying the model as a new set of rules named MLRs, short for *machine learning rules*. Packets targeted for inference follow the red arrows in Figure 2. When a packet arrives at the UPF and finds a PFCP session after the lookup, another lookup is performed for the first matching PDR. For ML inference, the MLR and its associated PDR will be prioritized for selection during the PFCP session's PDR lookup. When the MLR is selected, it executes the encoded ML model by applying a set of tables which match packet features and assign a traffic class *e.g.*, benign or malicious in binary classification, or a specific attack category.

The classification result is then added to the match key of the associated PDR which when matched invokes a set of specified instructions for further packet processing. This is analogous to what happens when an ordinary PDR assigns a traffic class to a packet leading to the selection of a FAR, BAR, QER and/or URR. In the case of the MLR which performs intrusion detection, its PDR could invoke a FAR which forwards benign traffic normally but drops malicious traffic or forwards it to a honeynet for further analysis. A full integration of MLRs as described above will lead to an ML-enabled UPF that can perform line-rate intrusion detection on live 5G traffic with high throughput and low latency.

B. ML model preparation

To realize a seamless integration of ML into the 5G UPF, the model must blend into the existing packet processing pipeline of the programmable-switch-based UPF. Several recent works have deployed ML models into programmable switches [14], [16], [17], FPGA-enhanced switches [26], SmartNICs [27], or split between switches and the control plane [28]. To enable real-time intrusion detection at full line rate on the programmable-switch-based 5G UPF, we opt for an in-switch deployment.

Model choice. Most fully in-switch inference solutions employ tree-based models like Decision Trees (DTs) and Random Forests (RFs) which have simple logical structures that make them easier to map to the switch match & action pipeline [14], [16], [17]. Moreover, DTs and RFs often outperform or are on par with more complex models like Neural Networks (NNs) when handling tabular data [29]. We therefore adopt DT/RF models for the UPF intrusion detection task and target inference on a per-packet basis as per normal UPF processing.

Owing to the strict constraints of programmable switch environments, all the steps involved with model training occur offline in an ML server [12]. The steps involved include feature extraction, feature selection and model training, and the mapping of the model to the switch match and action pipeline. These steps are described next.

Feature extraction. The datasets for most in-switch ML tasks typically come as raw packet captures in PCAP files. In this phase, we employ Tshark to extract the packet header fields which serve as features for per-packet inference. We extract 13 features from TCP, UDP and ICMP packets namely; source port, destination port, packet length, protocol, TCP flags (SYN, ACK, FIN, PSH, RST), TCP header length, TCP window size, UDP length, and time-to-live (TTL). When a packet does not have any of the listed features, it is set to zero. The extracted packets are then labelled using the labels provided by the dataset authors and saved in CSV files. Using only header fields ensures that privacy is preserved since we do not inspect packet payloads containing user data.

Feature selection and model training. The feature selection and model training process follows the approach adopted by previous work [17]. The process begins by training a large DT model using the Scikit-Learn Python libraries [30]. This model is trained using all 13 available features. The model's feature importance attribute is used to obtain feature importance and rank features according to their Mean Decrease in Impurity (MDI). Several new models are then trained and evaluated by adding the features one by one, starting with the most important, and saving the results for later analysis. To ensure that the final model will be hardware-feasible, we prune the generated trees to limit their maximum number of leaves to the upper limit of a TCAM match key. The maximum depth of the DT models is not limited and the trees are allowed to grow to full size and maximize their learning abilities. At the end of the modelling process, the best model is chosen based on a trade-off between accuracy and complexity.

Final model training and mapping to table entries. When the best model and its selected features are obtained, the final model is trained and converted to match and action table entries that will be loaded onto the switch by the controller. This process maps the model onto the switch pipeline using the tree mapping scheme originally proposed in Planter [25] and implemented in different flavours by other works [14], [16], [17]. In this mapping technique, the tree is broken down into feature tables, one per feature, and a code table. For each feature, we check all the nodes of the tree and extract all thresholds of the feature. The thresholds are then sorted in increasing order and ranges are constructed between every pair of thresholds. A code is assigned to each range and the concatenation of all codes assigned along the path to a tree leaf constitute a codeword. All the codewords and their corresponding classes are summarized on a code table which has as many entries as there are tree leaves. The feature tables are independent and can all be executed in a single match and action (M/A) stage *e.g.*, M/A 0 in Figure 3. The code table relies on the codes from the previous stage and so has to be executed in the next stage as also shown in Figure 3.

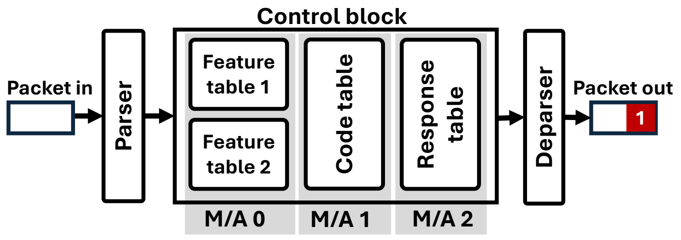

Fig. 3: In-switch workflow of the ML inference model

C. In-switch UPF model implementation

Building on Henna [17] and leveraging its open-source code, we write a P4 program to encode the trained model into the switch pipeline. We note that the final goal of this work would be to embed the intrusion detection model into a P4-based 5G UPF implementation *e.g.*, MacDavid *et al.* [22] or AccelUPF [21], leading to an ML-enabled UPF. However, as none of these existing implementations make their hardware code publicly available, this preliminary version focuses on describing our vision of how to integrate ML inference into the P4-based UPF, leaving the full integration with existing UPF implementations to future work. The in-switch per-packet inference solution is modelled according to the Protocol Independent Switch Architecture (PISA) which has three main parts; the parser, the match and action (M/A) pipeline, and the deparser, as shown in Figure 3. The various phases of the inference model are integrated into this architecture as follows.

Feature extraction in the parser. When packets arrive at the switch, they are first processed by the parser as shown in Figure 3, which extracts their header fields and saves them in metadata in the Packet Header Vector (PHV). These header fields constitute the features required for the inference task. They are carried by the PHV through the M/A pipeline where they are used for inference and any subsequent actions.

Packet classification. Once the MLR and its corresponding PDR are matched for the identified PFCP session, its features, *i.e.*, header fields, are retrieved from the PHV. They are then matched against the feature tables, represented in Figure 3 by Feature table 1 and Feature table 2, with each table assigning a code to the packet. The assigned codes are then concatenated to generate a codeword which represents the path from the root node to the leaf node where the packet arrives. The code table is then applied by matching the generated codeword to its corresponding leaf node to which a class is assigned *e.g.*, *class 1*, as in the outgoing packet in Figure 3.

Post-inference response. After a class is assigned to the packet, a set of instructions can be invoked in response to the classification result. This is depicted in Figure 3 as a response table which in the current implementation simply drops the packet if it is malicious or forwards it normally if it benign. The table matches the classification result and then mitigates the detected attack by dropping the packet or letting it go through if it is benign. This serves as a proof-of-concept to demonstrate how the switch-based UPF can respond to the ML-based inference results. More complex post-classification instructions such as BARs, URRs, and QERs can also be applied depending on the network settings.

Deparsing. When the packet is classified and a corresponding class-based response is assigned, the packet is repackaged in the deparser and processed according to the assigned rules.

IV. Experimental setup

We implement and evaluate our solution on a hardware testbed with Intel Tofino switches, using measurement data from a real-world test 5G network on a university campus.

A. Experimental platform

Our testbed is based on an Edgecore Wedge 100BF-QS switch, with the Intel Tofino BFN-T10-032Q chipset and 32 100-GbE QSFP28 ports. It serves as the P4-based UPF. We also have two DELL PowerEdge servers with Intel 8-core Xeon processors at 2 GHz frequency, 128 GB of RAM, and Mellanox ConnectX-5 NICs with 100 Gbps QSFP28 interfaces. We implement the model in the switch as a P4 program for per-packet inference as described in Section III-C. The first server injects user traffic from the test PCAP files to the switch using Tcpreplay [31]. After inference, the traffic is forwarded to the second server, where it is captured using Tcpdump and stored in a PCAP file for onward analysis. We also write a Python-based controller which also runs on one of the servers and configures the switch during the initial setup by loading the trained model to the switch in the form of M/A table entries and configuring switch ports and the the post-classification response table.

B. Use case: intrusion detection in a campus 5G network

To evaluate the proposed solution, an intrusion detection use case based on the 5G-NIDD dataset [32] is targeted.

Dataset description. It contains traffic generated on a testbed attached to the 5G Test Network (5GTN) at the University of Oulu in Finland. The data is collected from two base stations which each have several benign users and an attacker node. Benign traffic is captured from real users on the network. It includes traffic from various protocols such as HTTP, HTTPS, SSH, and SFTP. Live streaming services and web browsing generated the bulk of the HTTP and HTTPs traffic while SSH clients and servers deployed on mobile devices generated the SSH and SFTP traffic.

The attacker nodes target a server deployed in the 5GTN Multi-access Edge Computing (MEC) environment. There are two main attack scenarios: Denial of Service (DoS), and Port Scans. The goal of the DoS attacks is to slow down or completely shut down the target, making it inaccessible to legitimate users. Malicious content injection and flooding techniques are common methods for launching DoS attacks. The dataset contains five DoS attacks namely *ICMP Flood*, *UDP Flood*, *SYN Flood*, *HTTP Flood*, and *Slowrate DoS*.

Port scan attacks aim to identify opportunities for attacks through open ports. As such, they normally precede other attacks which exploit the discovered ports. Port scans generally send requests to a targeted range of ports in the victim host and wait for a response which it uses to determine the port status. There are three port scan attacks in the dataset namely *SYN Scan*, *TCP Connect Scan*, and *UDP Scan*. The goal of the classification task is to detect the eight different attacks and separate them from benign traffic.

Data pre-processing. The dataset comes in the CSV, ARGUS and PCAP formats for two base stations. For the PCAP format, the files are provided both with and without the GTP layer. The CSV files contain labelled files for each scenario which we use to generate labels for each 5-tuple (source IP, destination IP, source port, destination port, and protocol). As we do not implement the entire 5G network architecture, we use the PCAP files without the GTP layer. Then using Scapy, we split them into train and test PCAP files, using a train-test ratio of $75 - 25$. The 13 features listed in Section III-B are then extracted from the PCAPs using Tshark and Python and saved to labelled CSV files. We process the data for each base station separately before merging them. The feature selection and model training process described in Section III-B is then executed. The final DT model has a maximum depth of 37 and uses 10 features namely time-to-live, destination port, TCP window size, packet length, TCP Push flag, TCP header length, source port, TCP Reset flag, TCP Fin flag, and UDP length.

C. Performance metrics

The performance of the in-switch intrusion detection model is evaluated using common metrics derived from four main measures of a classification task, *i.e.*, true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). In this work, we express these metrics percentages to ease understanding. The metrics are computed as follows.

True positive rate (TPR) measures the number of positive samples that are rightly classified as positive *i.e.*, TP/(TP+FN).

False positive rate (FPR) is the fraction of negative samples wrongly classified as positive, computed as FP/(FP+TN).

True negative rate (TNR) quantifies the negative samples that are correctly classified as negative *i.e.*, TN/(TN+FP).

False negative rate (FNR) represents the portion of positive samples that are wrongly labelled negative *i.e.*, FN/(FN+TP).

F1 score is the harmonic mean of precision (TP/(TP+FP)) and recall (TP/(TP+FN)). It is widely used to express model performances in classification problems. It is computed using the formula 2TP/(2TP+FP+FN).

The above metrics are computed for each class and then averaged in two ways to obtain global values. The final value is either a *macro* average, *i.e.*, a simple average of individual class scores or a *weighted* average of class scores, weighted using the number of samples of each class. The macro score depicts the model performance irrespective of the representation of classes in the dataset, while the weighted score accounts for the imbalance in sample representation, and better represents the score of any random packet in the dataset.

V. Results and discussion

We evaluate the P4-based UPF intrusion detection model in terms of classification performance based on the metrics above and in terms of its consumption of key switch resources.

Class	F1 Score	TPR	FPR	TNR	FNR
Benign	99.993%	99.986%	0.000%	100.000%	0.014%
Slowrate DoS	90.507%	87.145%	0.663%	99.337%	12.855%
SYN Flood	100.000%	100.000%	0.000%	100.000%	0.000%
UDP Scan	99.727%	99.586%	0.000%	100.000%	0.414%
ICMP Flood	99.397%	100.000%	0.000%	100.000%	0.000%
TCP Connect Scan	99.589%	99.224%	0.000%	100.000%	0.776%
HTTP Flood	93.874%	96.292%	1.674%	98.326%	3.708%
SYN Scan	99.698%	99.891%	0.002%	99.998%	0.109%
UDP Flood	100.000%	100.000%	0.000%	100.000%	0.000%
Macro Avg	**98.087%**	**98.014%**	**0.260%**	**99.740%**	**1.986%**
Weighted Avg	**97.985%**	**97.998%**	**0.338%**	**99.662%**	**2.002%**

TABLE I: Classification performance of the in-switch model

A. Classification accuracy

The performance of the in-switch model is summarized in Table I. Overall, the model achieves 98.087% macro F1 score in attack classification, shedding light on how the in-switch DT model excels in separating the 8 attacks from benign traffic. Considering individual class F1 scores, the values range from $90.507\% - 100\%$, with most classes scoring above 99%. Only the Slowrate DoS and HTTP Flood attacks score less than 99%, *i.e.*, 90.507% and 93.874% respectively. The lower score for the Slowrate DoS attack is not entirely surprising since the slow rate masks the attack and makes it harder to identify. HTTP Flood has a better score than Slowrate DoS but its lower score relative to the other attacks could be attributed to the diverse nature of HTTP traffic coming from multiple web-browsing applications which renders the attack less identifiable, hence the lower score.

The above tendencies also apply to the TPR, where only Slowrate DoS and HTTP Flood have scores less than 99%. TNRs are also high, greater than 98% in all cases, with an overall value of over 99%. In terms of FPR, only HTTP Flood has a score of more than 1%, while Slowrate DoS has a score of 0.663%. All other classes have scores less than 0.005% as shown in Table I, confirming that the intrusion detection model does not wrongly classify a lot of benign traffic as malicious and/or classify one attack class as another. FNRs are also generally less than 0.5%, the only high ones being Slowrate DoS with 12.855%, and HTTP Flood with 3.708%. We also attribute these high values to same reasons we described above.

Overall, these results demonstrate that a fully in-switch model deployed in a P4-programmable-switch-based 5G UPF can perform attack detection and classification at line rate, achieving high accuracy. These are promising results which further motivate the concept of an ML-enabled UPF.

B. Resource consumption

We assess the switch memory consumption of the in-switch model using the Intel P4 Insight tool which provides a detailed view of the mapping of P4 programs onto switch hardware. The results are summarized in Table II which captures the consumption of six key resources. Looking at the top two key resources, *i.e.*, Static Random Access Memory (SRAM) and Ternary Content-Addressable Memory (TCAM), the model consumes just 1.4% and 8.0% respectively of the total available on the switch. This means our model is memory-efficient, leaving enough resources for cohabitation with other P4-based

SRAM	TCAM	Ternary Match Input Xbar	VLIW	Action Data Bus Bytes	Logical Table ID
1.40%	8.00%	11.50%	2.10%	4.60%	5.70%

TABLE II: Usage of switch resources by the in-switch model

UPF processes. For example, MacDavid *et al.* report that their hardware P4-based UPF pipeline consumes less than 15% of the Tofino chip's total available SRAM to support $17,500$ users. Thus, deploying our intrusion detection model alongside the complete P4-based UPF will increase SRAM consumption by only 9% of what the UPF implementation already requires, keeping total consumption at just over 16% of the total available on the hardware switch.

Ternary Match Input Crossbar (Xbar) consumption is relatively higher, at up to 11.5%. This is due to the model's use of 10 features and a large tree whose codewords require many Xbars to store the range and ternary match keys used respectively for the feature and code tables of the model. The Very Long Instruction Words (VLIW) enable parallelism and manage the execution of operations. The model consumes only 2% of VLIW which is quite modest. Lastly, we look at the Action Data Bus Bytes that transport action data across the switch pipeline and the Logical Table ID which are used for representing logical operations as M/A tables. Again, consumption of these resources is moderate, at just 4.6% and 5.7% respectively, leaving enough room for other functions.

In terms of packet processing latency, the in-switch model operates at full line rate, requiring less than 500 nanoseconds to both infer on a packet and forward it. This ultra-low latency combined with the low resource consumption further contribute to making the concept of an ML-enabled P4-based UPF an interesting prospect.

VI. Conclusions and future work

The main goal of this paper was to outline a proposal for an ML-enabled 5G UPF based on a P4-programmable switch that can enable real-time intrusion detection on live network traffic. We built on recent successes in deploying trained ML models into P4 switches and advancements in developing hardware-accelerated UPFs on similar devices to propose a workflow for introducing ML into the UPF as a special PDR. We then deployed a trained DT model into an off-the-shelf P4 switch and conducted experiments with data from a campus 5G test network. Our results showed how the in-switch model achieved high classification accuracy while running at line rate and keeping switch resource consumption at low levels.

This paper serves as a proof-of-concept on which future work could build to enable the full integration of ML models into new or existing P4-based 5G UPF implementations by identifying and addressing challenges arising from the proposed ML integration. We hope this paper will trigger research efforts in that direction and contribute to enabling a more seamless integration of ML inference into 5G user planes for real-time traffic analytics. We expect this work to also be relevant in the transition to 6G networks which are expected to have native support for AI.

Acknowledgments

A. Akem was supported by project PCI2022-133013 (ECOMOME), funded by MICIU/AEI/10.13039/501100011033 and the European Union "NextGenerationEU"/PRTR. M. Fiore was supported by NetSense grants no. 2023-5A/TIC-28944 funded by Comunidad de Madrid, and by the SNS JU and the European Union's Horizon Europe research and innovation program, under Grant Agreement No 101139270 (ORIGAMI).

References

[1] S. Fonyi. Overview of 5G security and vulnerabilities. *The Cyber Defense Review*, 5(1):117–134, 2020.
[2] Positive Technologies. 5G security issues, 2019.
[3] S. Khozam et al. DDoS mitigation while preserving QoS: A deep reinforcement learning-based approach. In *IEEE NetSoft*, 2024.
[4] A. Chetouane and K. Karoui. A survey of machine learning methods for DDoS threats detection against SDN. In *DiCES-N*, 2022.
[5] T. A. Tang et al. Deep learning approach for network intrusion detection in software defined networking. In *WINCOM*, pp. 258–263, 2016.
[6] K. He et al. Measuring control plane latency in SDN-enabled switches. In *SOSR*. ACM, 2015.
[7] Intel. Tofino Switch. https://shorturl.at/Ynzc6.
[8] NVIDIA. NVIDIA BlueField Networking Platform. https://www.nvidia.com/en-gb/networking/products/data-processing-unit/.
[9] P. Bosshart et al. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3), 2014.
[10] E. F. Kfoury et al. An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access*, 9:87094–87155, 2021.
[11] C. Zheng et al. In-network machine learning using programmable network devices: A survey. *IEEE Commun. Surv. Tutor.*, 2023.
[12] A. Sapio et al. In-network computation is a dumb idea whose time has come. In *ACM HotNets*, 2017.
[13] Z. Xiong and N. Zilberman. Do switches dream of machine learning? toward in-network classification. In *ACM HotNets*, 2019.
[14] A. T.-J. Akem et al. Flowrest: Practical flow-level inference in programmable switches with random forests. In *IEEE INFOCOM*, 2023.
[15] A. T.-J. Akem et al. Encrypted traffic classification at line rate in programmable switches with machine learning. In *IEEE NOMS*, 2024.
[16] G. Zhou et al. An efficient design of intelligent network data plane. In *USENIX Security*, 2023.
[17] A. T.-J. Akem et al. Henna: hierarchical machine learning inference in programmable switches. In *NativeNI*, 2022.
[18] K. Du et al. Definition and evaluation of latency in 5G: A framework approach. In *IEEE 5GWF*, 2019.
[19] ETSI. LTE; 5G; interface between the control plane and the user plane nodes (3GPP TS 29.244 version 16.5.0 release 16), 2020.
[20] G. Amponis et al. Threatening the 5G core via PFCP DoS attacks: the case of blocking UAV communications. *J. Wirel. Commun. Netw.*, 2022.
[21] A. Bose et al. AccelUPF: accelerating the 5G user plane using programmable hardware. In *SOSR*. ACM, 2022.
[22] R. MacDavid et al. A P4-based 5G user plane function. In *SOSR*, 2021.
[23] Z. Cong et al. CeUPF: Offloading 5G user plane function to programmable hardware base on co-existence architecture. In *ICEA*, 2022.
[24] A. T.-J. Akem et al. Jewel: Resource-efficient joint packet and flow level inference in programmable switches. In *IEEE INFOCOM*, 2024.
[25] C. Zheng and N. Zilberman. Planter: Seeding trees within switches. In *SIGCOMM*, 2021.
[26] T. Swamy et al. Taurus: a data plane architecture for per-packet ML. In *ASPLOS*. ACM, 2022.
[27] G. Siracusano et al. Re-architecting traffic analysis with neural network interface cards. In *USENIX NSDI*, 2022.
[28] D. Barradas et al. Flowlens: Enabling efficient flow classification for ML-based network security applications. *NDSS*, 2021.
[29] L. Grinsztajn et al. Why do tree-based models still outperform deep learning on typical tabular data? In S. Koyejo et al., editors, *Advances in Neural Information Processing Systems*, volume 35, 2022.
[30] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2011.
[31] A. Turner and F. Klassen. Tcpreplay, 2013.
[32] S. Samarakoon et al. 5G-NIDD: A comprehensive network intrusion detection dataset generated over 5G wireless network, 2022.