A Framework for In-network Inference using P4

Huu Nghia Nguyen, Manh-Dung Nguyen, Edgardo Montes de Oca
firstname.lastname@montimage.com
Montimage
Paris, France

ABSTRACT

Machine Learning (ML) has been widely used in network security monitoring. Although, its application to data intensive use cases and those requiring ultra-low latency remains challenging. This is due to the large amounts of network data and the need of transferring data to a central location hosting analysis services. In this paper, we present a framework to perform in-network analysis by offloading ML inference tasks from end servers to P4-capable programmable network devices. This helps reduce transfer latency and, thus, allows faster attack detection and mitigation. It also improves privacy since the data is processed at the networking devices. The paper also presents an experimental use-case of the framework to classify network traffic, and to early detect and rapidly mitigate against IoT malicious traffic.

KEYWORDS

In-network inference, programmable network, open-source, P4, attack detection, IoT networks

ACM Reference Format:
Huu Nghia Nguyen, Manh-Dung Nguyen, Edgardo Montes de Oca . 2024. A Framework for In-network Inference using P4. In *The 19th International Conference on Availability, Reliability and Security (ARES 2024), July 30-August 2, 2024, Vienna, Austria.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3664476.3670453

1 INTRODUCTION

Recently, ML approaches have been successfully applied in different domains and have shown significant breakthroughs. They have been applied to different applications in networking, ranging from traffic classification and anomaly detection to network configuration and orchestration. A ML-based application usually consists of 2 main steps: model preparation and inference. The former consists of preparing data, training and optimising ML models. The latter deploys the obtained models to production to receive input data, make predictions, and return results.

Furthermore, Software Defined Networking (SDN) is pushing an emerging approach to network management by decoupling control and data planes. SDN enables greater automation, flexibility, and scalability in network management. It helps to reduce the need of dedicated hardware devices by introducing programmable devices to process and control network traffic. It allows network administrators to centrally manage and program network behavior via a software-based controller, such as managing the flow of data traffic (data plane) and making decisions about how it should be forwarded based on network policies and configurations.

However, only the control plane has been programmable in SDN. The data plane was not programmable in a flexible way, and allowed only the pre-configured policies and configurations. Applying ML to data intensive networking use cases and those requiring ultra-low latency is challenging because the inference step is performed at a central location hosting analyses services in the control plane. This requires transferring data extracted from the data plane. The prediction results are then used to reprogram the data plane via the controller. This Round-Trip Time (RTT) causes delays in the closed-loop detection and mitigation process. To improve this, there is a need to perform ML inference at the data plane.

Programming Protocol-independent Packet Processors (P4) [3] is a domain-specific language for network devices, specifying how data plane is to be processed by networking devices, such as, switches, routers, filters, etc. Thus, it opens new possibilities for flexible and dynamic programmable data planes, such as, performing the ML inference. Offloading ML inference from the control plane side to the data plane side helps not only reducing RTT delay in the closed-loop but also avoiding privacy issues because data is processed only by the the data plane devices.

This paper presents a complete open-source framework to perform ML-based inference at the data-plane using P4. Especially, we make the following contributions:

- We extend our existing tool, Montimage AI Platform (MAIP) [12], to deal with Decision Tree (DT) models. MAIP provides users with easy access, through a friendly and intuitive user interface, to prepare ML models and understand the models using Explainable Artificial Intelligence (XAI). We also enhance it to give users the ability to select features to be extracted from .pcap files, and to tune parameters for training models.
- We propose a new transformation of a DT model to a single Match-Action (MA) table to reduce the number of MA entries and table lookup.
- We implement the framework[1] and demonstrate its application via a network traffic classification use-case. The use-case is deployed in a Raspberry Pi to act as a smart IoT gateway to be able to swiftly detect and react against malicious IoT encrypted traffic.

This paper is structured as follows. Section 2 discusses the state of the art and the related work. Section 3 describes the proposed framework and its detailed components. An application user-case of

[1]The framework is freely available at https://github.com/montimage/maip and https://github.com/montimage/in-network-inference-using-p4

the framework will be introduced in Section 4. Section 5 concludes the paper.

2 RELATED WORK

We summarize in this section two main approaches related to our framework: XAI and offloading ML inference to programmable data planes. The former helps to better understand trained ML models, thus easily optimise them. The latter helps to more rapidly detect anomalies and mitigate them.

2.1 Explainable AI

XAI [2] aims to make Artificial Intelligence (AI)/ML black-box models more transparent by explaining why decisions are made. Across various domains, AI/ML plays a crucial role, but trust and transparency are essential for its future applications. These models are usually complex and not easily interpretable due to their multiple layers and hyperparameters. This complexity hinders both users and developers from understanding and improving their accuracy and performance. Therefore, incorporating explainability on top of these models is necessary to provide post-hoc explanations and enhance interpretability. Notable post-hoc explainability methods include visual explanations, local explanations, explanations by example, and feature relevance explanations.

Local explanations seek to approximate explanations within less complex solution sub-spaces by considering only a subset of data. One popular technique is Local Interpretable Model-agnostic Explanations (LIME) [15], which interprets outputs of black-box models across various fields.

Feature relevance explanations involve computing relevance scores for model features to quantify their contribution or sensitivity to the model's output. Shapley Additive Explanations (SHAP) [10] is a popular XAI technique utilizing cooperative game theory to identify the importance of each feature value in a prediction. Additionally, *Permutation Feature Importance* is a global XAI method that measures changes in prediction error when feature values are randomly permuted.

Explanations by example focus on extracting representative data examples that relate to a model's generated result, thereby enhancing understanding. Methods within this category include counterfactual explanations [18] and adversarial examples.

2.2 Offloading ML Inference to Programmable Data Planes

Data plane programmability allows the network owner to define data plane functionality using software artifacts running on programmable networking devices. P4 [3] is a domain-specific language used for programming these devices to process packets. A key feature of the P4 language is protocol independence. It supports flexible interaction between the programmable data plane and control plane, which enables coordination between the control logic and packet processing logic on devices.

Although a P4 program is independent from a device running it, its compilation needs to follow the packet processing architecture of the device. This architecture defines the high level structure of the device, and the interfaces between its major components. The most common architecture is the Protocol Independent Switch Architecture (PISA), which generalizes the Reconfigurable Match-Table (RMT) [4] model and provides essential line-rate packet processing features. In the PISA architecture, packets go through a packet parser, which instantiates user-defined protocols. After the parser processes a packet, it follows a pipeline of control flows and MA tables. Finally, packet headers are emitted by a deparser.

MA table is the core mechanism for processing packets [3]. It's basically a hash lookup table, in which an entry consists of a key to match against an input and a value is an action to be executed. MA table is usually used to define a set of rules to check the packet's header fields against a set of predefined criteria. The matching criteria can be `exact`, `range`, `lpm` and `ternary` to match exactly a given value, a given range of values, a longest prefix and a ternary respectively. If the packet matches a rule, an Action is taken, such as, modifying the packet's headers, copying the packet, dropping the packet, forwarding it to a particular port, or any other defined operations that can be applied to the packet within the switch. Users can use Actions to implement different logic, such as, performing Distributed Denial of Service (DDoS) attack detection and reaction at the edge [13], performing in-band network telemetry [11], implementing a Time Sensitive Networking (TSN) mechanism [8], or even performing in-network ML inference [1, 7, 19, 20] which will be subsequently detailed.

In-network ML inference refers to the process of offloading ML inference to networking devices [21]. It provides line rate ML inference on programmable network devices within the network. This is different from traditional ML services that train and deploy models either on a server or an accelerator, e.g., GPU, using complex frameworks such as, Sklearn[2], TensorFlow[3]. Here, in-network ML first trains a model on a server at the control plane, then translates the model to a set of MA entries to define packet processing logic, and finally loads the entries to do inference on a network device, at the data plane.

The authors in [19] present algorithms, called IISY, to transform different ML models, such as, Decision Tree, SVM, K-mean, Naïve Bayes. The authors also propose an evaluation prototype of the algorithms. IISY then follow different approaches to apply in-network ML inference to different ML methods, such as, supervised, unsupervised, reinforcement, or distributed learning. Two surveys [14, 21] summarise these approaches.

The existing approaches mainly focus on tree-based ML models, such as, DT or Random Forest (RF) due to their simple logical structure and limited operations involved at networking devices [1]. SwitchTree [7] extends [19] and deals with RF models by using `range` to match a range of values, instead of `exact` as in IISY, to reduce number of MA entries. Although, these approaches need $n+1$ MA tables to map a tree of n features. Our transformation requires only a single MA table, thus reducing the number of pipelines to be executed when performing table lookup. The authors in [20] focus on implementing a mechanism of seamless updates of in-network ML inference models at runtime. The authors in [1] deal with the challenges and experiment the in-network ML inference on a P4-enabled hardware switch.

[2]https://scikit-learn.org
[3]https://www.tensorflow.org

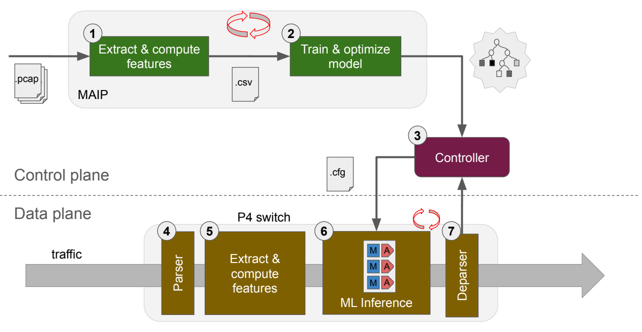
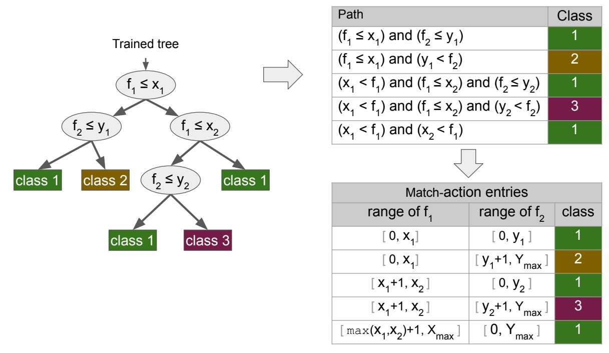


Figure 1: Overview of the framework



Path	Class
$(f_1 \le x_1)$ and $(f_2 \le y_1)$	1
$(f_1 \le x_1)$ and $(y_1 < f_2)$	2
$(x_1 < f_1)$ and $(f_1 \le x_2)$ and $(f_2 \le y_2)$	1
$(x_1 < f_1)$ and $(f_1 \le x_2)$ and $(y_2 < f_2)$	3
$(x_1 < f_1)$ and $(x_2 < f_1)$	1

Match-action entries		
range of f_1	range of f_2	class
$[0, x_1]$	$[0, y_1]$	1
$[0, x_1]$	$[y_1+1, Y_{max}]$	2
$[x_1+1, x_2]$	$[0, y_2]$	1
$[x_1+1, x_2]$	$[y_2+1, Y_{max}]$	3
$[\mathtt{max}(x_1,x_2)+1, X_{max}]$	$[0, Y_{max}]$	1

Figure 2: Example of generation of Match-action entries

3 SYSTEM DESIGN

Figure 1 represents an overview of the framework and its components. The framework consists mainly of two parts, model preparation and in-network ML inference. The model preparation, including blocks ① and ②, is done offline at the control plane. The in-network ML inference, blocks ④,⑤,⑥, and ⑦, is done online at a P4-enabled switch to verify network traffic against the model. The communication between the control plane and the data plane is done via a controller in ③. We will detail these blocks subsequently.

3.1 Offline Model Preparation

The blocks ① and ② show the extract feature process and model training process, respectively. We follow the standard ML pipeline and implement the two processes in MAIP [12], which is an open source ML-based framework for anomaly detection in encrypted traffic with high performance, explanation and robustness against adversarial attacks. It also provides an intuitive and user-friendly interface to access a range of ML services, including feature extraction, model building and storing, adversarial attack injection, explanation generation, and AI models evaluation using different quantifiable metrics.

The block ① employs our open source tool MMT-Probe[4] to parse raw network traffic in .pcap files, extract the needed information, compute the features required for training ML models, and, then translate them into a numeric form in Comma-Separated Values (CSV) format. Specifically, MMT-Probe is a monitoring and data extraction software that parses network traffic to extract network and application-based events, such as protocol field values and statistics. It allows parsing a variety of network protocols, e.g., TCP, UDP, HTTP, and more than 700 others, for the purpose of extracting metadata. The features consist of multiple parameters that can be directly extracted from raw traffic, such as, IP addresses, packet size, etc, or calculated from the extracted values, such as, variation of packet sizes, Inter-Arrival Time (IAT) showing the time intervals between successive arrivals of packets or events, etc.

The block ② aims to train, fine-tune, and optimize ML models in a closed-loop manner. It employs popular XAI methods such as SHAP to identify a list of the most important features contributing to the models' predictions. Based on post-hoc insights provided by XAI methods, we can retrain the models using only the important features and discard those that did not contribute much to the models' outcomes. We can iterate through this loop several times to fine-tune the process of blocks ① and ② until the model's accuracy is significantly improved. Furthermore, we provide users with the option to select which features they want to use for training models based on their domain-specific knowledge. The output of these two blocks, ① and ②, is a highly accurate trained model that will be ready for subsequent analysis in the following blocks.

3.2 Model Transformation

The block ③ in Figure 1 is implemented by a controller. The controller inputs a ML model, transforms it into a set of MA entries, then loads them into the switch. It can also receive ML inference results from the switch. We currently focus on supporting Decision Tree models.

The transformation is performed by visiting all possible paths from the root to a leaf node of the tree. Each path is transformed into a MA entry. The Match of an entry is a set of ranges of features' values that satisfy the path. The Action of an entry is to return the classification result.

Figure 2 demonstrates a simplified transformation of a DT with two features f_1 and f_2. A DT is a binary tree, in which, each node is a simple boolean expression that represents a relation between a feature and a threshold. If the feature's value is less than or equal to the threshold, then the left branch is taken, otherwise it's the right branch. In the DT tree in the left side of Figure 2, we use $x_i \in [0, X_{max}]$ and $y_i \in [0, Y_{max}]$ to denote the feature thresholds f_1 and f_2 respectively. These thresholds are literal numeric values.

The ML inference process, which predicts a set of feature values f_1, f_2 against this DT tree, is basically the verification of f_1 and f_2 against these boolean expressions from the root to a leaf node. The leaf contains the prediction. For example, if we have $(f_1 \le x_1) \wedge (f_2 \le y_1)$ then the prediction is class 1.

We list all possible paths of the tree on the table on the top-right of Figure 2. Because x_i and y_i are literal numeric values, we can easily collapse the boolean expression of a path to ranges of possible values of each feature. The bottom-right table of Figure 2 represents the result MA entries. Each entry is a row of the table. For example, the first row represents an entry having the key (Match)

[4]https://github.com/Montimage/mmt-probe

corresponding to range values f_1, f_2: $[0, x_1]$ and $[0, y_1]$, and value (Action) that is class 1.

Previous approaches in [7, 19, 20] transform a DT tree as-is, i.e., without collapsing. [7, 19] follow the sequential top-down tree path to map the each threshold to a MA entry. The thresholds (thus its MA entries) are then grouped by feature into a separated table. They then introduces another table to combine the mapping results from the feature tables to the final result. This map is enhanced in [20] to reduce number of entries in each MA table by breaking the sequential dependency of each node in a path. However, these approaches require $n + 1$ tables for a DT tree of n features, and more tables implies more pipeline executions. By simply collapsing the conjunction of a boolean expression of a path before mapping, our transformation produces a single MA table for a DT input within a minimum number of entries. Indeed, the number of entries is less than or equal to the number of leaf nodes. Consequently, this reduces the number of pipeline executions needed to match features.

3.3 In-network ML Inference

The blocks ④, ⑤, ⑥ and ⑦ are implemented inside a P4-enable switch. The blocks are described in the following.

Parser. When a packet arrives at the switch, it is parsed in block ④. This parser extracts the concerned protocol headers, such as, Ethernet, IP, UDP, etc. The extracted values are kept as metadata in the Paacket Header Vector (PHV). The values can then be subsequently accessed by other blocks. For example, Ethernet and IP protocols need to be parsed to be able to perform packet switching.

In-band features extraction & computation. The block ⑤ in Figure 1 extracts the necessary feature values required for ML inference. Several feature values can be obtained from the previous parser block, such as, the length of an IP packet, the source port number, etc. Although, there are feature values that still need to be extracted or computed, for example, IAT feature that captures the different arrival times of two consecutive packets, or the maximum size of packets during a window time frame, etc. This extraction and computation must take into account a number of strict constraints imposed by a programmable switch, such as, low available memory, limited support for mathematical operations and limited number of operations per packet to maintain line rate packet processing.

ML inference. This block performs ML inference. It predicts the set of extracted feature values. The prediction is done by matching the values against the MA entries. If there exists an entry having a key that matches the values then the prediction is the result value. This result is saved as metadata so that it can be accessed by the next block.

Deparser. This is the last block in the chain of processing the packets inside the switch. This block, ⑦, packages a packet with additional information, such as, MAC destination, new checksum, etc, and then sends the packet to a selected outgoing port of the switch. This block also notifies the controller on the result obtained from the ML inference. Additional information is also sent to the controller, such as, a 5-tuple identifying the packet (i.e., source and destination IP addresses and port numbers, and protocol identification), as well as the feature values. The information is encoded in a digest message (i.e., a message to communicate information from the data plane to the control plane), then sent to the controller. Utilisation of digest communication, instead of transmission of a packet to the controller via the CPU port, reduces the processing overhead at the controller [3] as the message is structured. Digests are sent to the controller by calling `digest` P4 function. The controller is configured with P4Runtime [3] to listen to digest messages. Upon receiving a message, the controller decodes it. The decoded information can be used to perform some reaction, such as, to reconfigure the switch, or even to retrain the model, etc. This utilisation is out of scope of the paper, thus here, for instance, we simply save it into a .csv file.

Table 1: Overview of the dataset

	Number of packets	Label
Normal traffic	155888	0
Malicious traffic	10208	1
Total	166096	

It is crucial to note that the blocks presented above are essential. Depending on a specific use-case, other blocks can be introduced. For example, a flow tracker block can be inserted after the parser block to track the flows that have already been classified, or identified as malware. If so, any new incoming packets belonging to those flows are not processed by the blocks ⑤ and ⑥, but are forwarded as-is or dropped respectively.

The prediction result can also be immediately used by the switch, for example, to decide to drop or forward the current packet. This forms a local closed-loop of extraction-detection-reaction inside the switch. Consequently, it avoids RTT delay caused by the communication with the controller.

4 EXPERIMENTAL EVALUATION

In this section, we evaluate our framework by applying it to implement a smart IoT wireless gateway. The gateway implements an in-network ML-based solution for quickly detecting and immediately mitigating IoT malicious traffic which is encrypted. The threat model simply follows the block-list (or blacklist) model. This is to block any packets which are classified as malicious.

4.1 Model Preparation

Several packet-level features [12] can be easily extracted from packet header fields and utilized to address various traffic classification challenges. However, they may prove inadequate when dealing with encrypted traffic, as certain features can be obfuscated by encryption algorithms [6]. Therefore, for the sake of simplicity, in this experimental evaluation we focus solely on statistical features derived from packet sizes and timestamps. Specifically, we consider three key features: *IAT*, representing the inter-arrival time between packets; *len*, indicating the payload size of each IP packet; and *diffLen*, which captures the variability in packet sizes over time.

Table 1 presents a summary of the public dataset CSE-CIC-IDS2018 [16], which is used for botnet detection in encrypted traffic within IoT networks. The dataset contains a total of 166096 packets

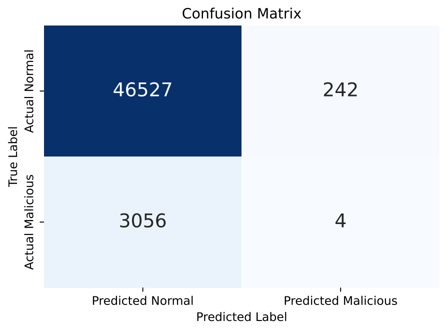


Figure 3: Confusion matrix

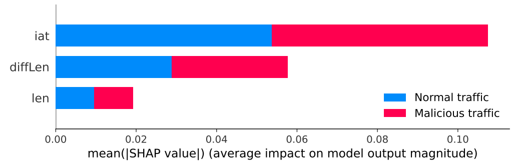


Figure 4: SHAP summary plot for anomaly detection

extracted from .pcap files, with 155,888 packets classified as normal traffic and 10208 packets classified as malicious traffic. In the dataset, normal and malicious traffic are labeled as 0 and 1, respectively. We use MAIP to extract the 3 features from this dataset. We then randomly spit the obtained values into training and testing datasets. The training dataset consisting of 70% of the data is used to train our DT model. The rest is used to test the accuracy of obtained model.

Figure 4 shows the confusion matrix that provides a detailed breakdown of the accuracy of the obtained DT model. For instance, the top-left cell represents instances where the model correctly classified normal traffic (label 0) as normal, with a count of 46527. The accuracy metric, calculated as the ratio of correctly predicted instances to the total number of instances, is 93.38%. Despite their simple logical structure and lower precision compared to advanced ML techniques such as deep neural networks [12], the DT model still achieves high accuracy and, more importantly, facilitates the complex operations involved in networking devices.

SHAP provides explanations of a model's predictions by identifying the most important features based on a feature attribution framework and Shapley values. Figure 4 illustrates important features by sorting the sum of magnitudes of Shapley values over these samples. Here, the length of the bar indicates how much influence the feature has on the prediction. Among the three features used, the most important one is *IAT*. It is also a common characteristic used in machine learning algorithms [5, 17], as malicious communications often exhibit specific flow duration patterns. For instance, some botnets establish brief connections, while others are more chatty, resulting in longer duration. Please note that the efficacy of this detection method may diminish due to attacker evasion tactics, though this aspect falls beyond the scope of this paper.

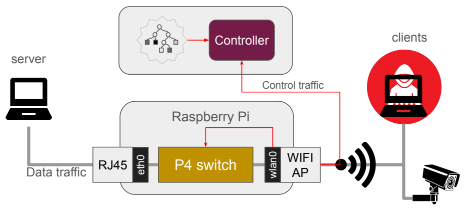


Figure 5: Overview of the IoT testbed

4.2 In-network ML Inference

The framework prototype is implemented mainly in P4 and Python to provide the data plane and control plane functionality (see Figure 1). We execute the P4 code in a P4 software switch which implements the behavioral model version 2 architecture, BMv2[5]. The DT model, that is prepared in the previous section, is transformed into 5022 MA entries of a table that will be loaded into the BMv2 P4 switch. With the same model, IISY [19] generates totally 10047 MA entries in 4 MA tables. Three tables for *IAT*, *len* and *diffLen* features contain 4731, 42 and 251MA entries respectively. The last table contains 5023 MA entries for synthesising final results from the 3 tables above.

Correctness of P4-based Inference. We first want to evaluate the correctness of the ML inference running inside the P4 switch. We use mininet[6] to create a realistic virtual network running on a single virtual machine Ubuntu 20.04.6. The network consists of two nodes: a host h and a BMv2 P4 switch s. The captured packets in dataset traces are replayed from h to s using tcpreplay[7]. For each incoming packet, the P4 switch performs a prediction, e.g., steps ④, ⑤, ⑥ and ⑦ in Figure 1, and sends the prediction result together with feature values to the controller which saves this information into a .csv file. We later use sklearn to obtain the score which represents the mean accuracy of data in the .csv file. Since we obtained the score 1, we can conclude that the P4-based inference and sklearn-based inference give the same result.

Overhead on Packet Latency. In this section, we evaluate the overhead caused by the in-network ML inference over packet latency in a physical testbed. We deployed the implementation prototype of the in-network ML inference in Raspberry PI 3 Model B, with 1GB of RAM and Quad Core 1.2GHz, which acts as a smart IoT wireless gateway, as shown in Figure 5. We rely on P4PI [9] to run the P4 code on a BMv2 P4 switch in the Raspberry Pi. The controller is deployed on a separated machine. The controller and other IoT devices connect to the gateway via its wireless network interface. The IoT devices can connect to each other and to a server represented on the left side of the figure.

In order to assess the latency overhead, we developed a pair of basic client and server applications to actively gauge the end-to-end packet latency. The client includes its current time in a packet payload and transmits it to the server, which promptly returns the packet. The client subsequently contrasts the current time with the one encapsulated in the packet to determine the RTT of the

[5]https://github.com/p4lang/behavioral-model/blob/main/docs/simple_switch.md
[6]https://mininet.org/
[7]https://tcpreplay.appneta.com/

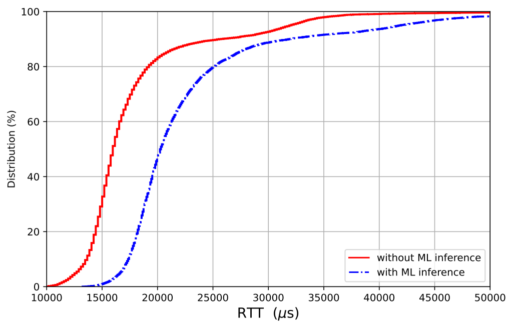


Figure 6: Overhead of ML inference on packet latency

packet, all of this is achieved without requiring time synchronization between the client and the server. In the case of measurements without ML inference, we removed the related P4 code of ML in the P4 switch.

We conducted several measurements. Each measurement sends 10000 packets. We present the results in the Cumulative Distribution Function (CDF) diagram in Figure 6. The horizontal axis stands for the measured RTT values and the vertical axis for their distribution. We can see that almost all RTT values vary from 15000 to 30000 μs. The average latency with and without ML inference are 22093 μs and 19069 μs respectively. Therefore, the additional latency increases by 15.8%. This additional latency is mainly due to the table lookup time of the P4 switch that is executed inside the Raspberry Pi.

Swift Detection and Reaction at the IoT Gateway. We involved the testbed in Figure 5 by introducing a new block in the P4 program to drop a packet if it is classified as malicious. We then use a laptop, that connects to the gateway via its wireless interface, to act as a malicious IoT device, as shown in the right side of Figure 5. In this laptop, we use tcpreplay to inject botnet traffic into the network. We see that the gateway can detect almost all the malicious packets and immediately drop them. Although, there exist packets that are not classified as malicious because the accuracy of the model is 93.38%.

5 CONCLUSION

We presented in this paper a comprehensive framework to swiftly detect and mitigate malicious traffic by directly performing ML inference at the data planes via P4-enabled switches. We implemented the framework and experimentally evaluated it against a P4 software switch. This is a step forward to applying ML to security analysis of network traffic at line rate. Future work will explore improving the accuracy of the model and applying it for encrypted network traffic classification and anomaly detection in deterministic networks.

ACKNOWLEDGMENTS

This work is partially supported by the European Union's Horizon Europe research and innovation program under grant agreements Numbers 101096504 (DETERMINISTIC6G), 101070450 (AI4CYBER) and the INFLUENCE project. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

[1] Aristide Tanyi-jong Akem, Guillaume Fraysse, and Marco Fiore. 2024. Encrypted Traffic Classification at Line Rate in Programmable Switches with Machine Learning. In *Proc. of NOMS.*
[2] Alejandro Barredo Arrieta et al. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion* (2020).
[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. 2014. P4: Programming Protocol-Independent Packet Processors. *Computer Communication Review* 44, 3 (2014), 87–95.
[4] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In *Proc. of SIGCOMM.*
[5] Livadas Carl, R Walsh, D Lapsley, and WT Strayer. 2006. Using machine learning technliques to identify botnet traffic. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on. IEEE.*
[6] Hossein Doroud, Ahmad Alaswad, and Falko Dressler. 2022. Encrypted Traffic Detection: Beyond the Port Number Era. In *2022 IEEE 47th Conference on Local Computer Networks (LCN).* 198–204.
[7] Jong hyouk Lee and Kamal Sigh. 2020. SwitchTree: In-network Computing and Traffic Analyses with Random Forests. *Neural Computing and Applications* (2020).
[8] Fabian Ihle, Steffen Lindner, and Michael Menth. 2023. P4-PSFP: P4-Based Per-Stream Filtering and Policing for Time-Sensitive Networking. (2023).
[9] Sándor Laki, Radostin Stoyanov, Dávid Kis, Robert Soulé, Péter Vörös, and Noa Zilberman. 2021. P4Pi: P4 on Raspberry Pi for networking education. *SIGCOMM Comput. Commun. Rev.* 51, 3 (jul 2021), 17–21.
[10] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).
[11] Huu Nghia Nguyen, Bertrand Mathieu, Marius Letourneau, and Guillaume Doyen. 2023. A Comprehensive P4-based Monitoring Framework for L4S leveraging In-band Network Telemetry. In *Proc. of NOMS.*
[12] Manh-Dung Nguyen, Anis Bouaziz, Valeria Valdes, Ana Rosa Cavalli, Wissam Mallouli, and Edgardo Montes De Oca. 2023. A deep learning anomaly detection framework with explainability and robustness. In *Proceedings of the 18th International Conference on Availability, Reliability and Security (ARES '23).*
[13] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi. 2019. P4 edge node enabling stateful traffic engineering and cyber security. *Journal of Optical Communications and Networking* 11, 1 (2019), A94–A95.
[14] Ricardo Parizotto and Israat Haque. 2024. Offloading Machine Learning to Programmable Data Planes: A Systematic Survey. January 2024 (2024).
[15] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should I trust you?" Explaining the predictions of any classifier. In *Proc. of SIGKDD.* 1135–1144.
[16] Iman Sharafaldin, Arash Habibi Lashkari, Ali A Ghorbani, et al. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *Proc. of ICISSP* 1, 108–116.
[17] W Timothy Strayer, David E Lapsley, Robert Walsh, and Carl Livadas. 2008. Botnet detection based on network behavior. *Botnet detection* 36, August (2008), 1–24.
[18] Sandra Wachter, Brent Mittelstadt, and Chris Russell. 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.* 31 (2017), 841.
[19] Z Xiong and N Zilberman. 2019. Do Switches Dream of Machine Learning? Toward In-Network Classification. *Proc. of HotNets,* 25–33.
[20] Mingyuan Zang, Changgang Zheng, Lars Dittmann, and Noa Zilberman. 2023. Towards Continuous Threat Defense: In-Network Traffic Analysis for IoT Gateways. *IEEE Internet of Things Journal* 11, 6 (2023), 9244–9257.
[21] Changgang Zheng, Damu Ding, Shay Vargaftik, and Yaniv Ben-itzhak. 2023. In-Network Machine Learning Using Programmable Network Devices : A Survey. *EEE Communications Surveys & Tutorials* (2023), 1–35.