This is the author's accepted version of the article. The final version published by IEEE is A.T-J. Akem, G. Fraysse and M. Fiore, "Encrypted Traffic Classification at Line Rate in Programmable Switches with Machine Learning," NOMS 2024 - IEEE/IFIP Network Operations and Management Symposium, 2024, pp. 1-9, doi: 10.1109/NOMS59830.2024.10575394.

Encrypted Traffic Classification at Line Rate in Programmable Switches with Machine Learning

Aristide Tanyi-Jong Akem*†, Guillaume Fraysse‡ and Marco Fiore*
*IMDEA Networks Institute, Spain, †Universidad Carlos III de Madrid, Spain, ‡Orange Innovation Networks, France
{aristide.akem, marco.fiore}@imdea.org, guillaume.fraysse@orange.com

***Abstract*—Encrypted Traffic Classification (ETC) has become an important area of research with Machine Learning (ML) methods being the state-of-the-art. However, most existing solutions either rely on offline ETC based on collected network data or on online ETC with models running in the control plane of Software-Defined Networks (SDN), all of which do not run at line rate and would not meet latency requirements of time-sensitive applications in modern networks. This work leverages recent advances in data plane programmability to achieve real-time ETC in programmable switches at line rate, with high throughput and low latency. The proposed solution comprises (i) an ETC-aware Random Forest (RF) modelling process where only features based on packet size and packet arrival times are used, and (ii) an encoding of the trained RF model into production-grade P4-programmable switches. The performance of the proposed in-switch ETC framework is evaluated using 3 encrypted traffic datasets with experiments in a real-world testbed with Intel Tofino switches, in the presence of background traffic at 40 Gbps. Results show how the solution achieves high classification accuracy of up to 95%, with sub-microsecond delay, while consuming on average less than 10% of total available switch hardware resources.**

***Index Terms*—Encrypted traffic classification, machine learning, programmable switch, P4, random forest**

I. Introduction

With network systems' growing size and complexity, network traffic is increasingly being encrypted to cope with the growing need for information security. For example, as of September 2023, over 95% of Google traffic was encrypted [1]. Classifying encrypted traffic is important to operators for automating several tasks such as intrusion and anomaly detection, quality of service and quality of experience prediction, and routing optimization. Encrypted Traffic Classification (ETC) techniques have evolved over the years from simple port-based techniques to Deep Packet Inspection (DPI), traditional Machine Learning (ML), and more recently to Deep Learning (DL) techniques [2], [3].

In most cases, the ETC models are run entirely offline, trained on historical network traffic data and used to perform inference on traffic information gathered from network probes [4]–[10]. However, these models neither run in real-time nor at line rate. Other works run ETC models in the control plane of software-defined networks (SDN) [11]. While some of these works support real-time ETC [12]–[14], many others do not [15]–[17]. In both cases, they do not run at line rate. Some make attempts to accelerate the traffic classification process by running the models in SDN home gateways [13], [14]. However, these gateways are CPU-based and do not attain the line rate that is desired for very low latency applications.

The recent advancements in data plane programmability with the introduction of languages like P4 [18] and commercial programmable data planes like Intel Tofino [19] have sparked an interest in data plane inference. Deploying ML models for inference in the data plane promises higher throughput and lower latency inference. However, such deployment could be particularly challenging notably in the case of switches with strict constraints in terms of memory, support for mathematical operations, and the number of allowed operations per-packet.

Recent works have demonstrated the feasibility of in-switch machine learning inference with Decision Tree (DT) and Random Forest (RF) models for classifying traffic at line rate in programmable switches [20]–[24]. However, none of these works specifically focuses on ETC, neither do they take into account the possible unavailability of some features due to encryption. As such, to the best of our knowledge, *there is no existing specialized solution for running ETC entirely in programmable switches at line rate with high throughput and low latency.*

This work fills the above gap by building on recent successes in user-plane inference to integrate the ETC process fully into programmable switches to perform line rate inference on traffic flows in real-time using Random Forest (RF) models trained offline and then embedded into the switch pipeline. By so doing, the following contributions are made.

- A fully in-switch solution for ETC in the data plane is proposed. This enables line rate inference on traffic flows with high throughput and sub-microsecond latency.
- The solution involves ETC-aware RF models which only use statistical features based on packet sizes and packet Inter-Arrival Times (IATs), avoiding any features whose relevance could be impacted by encryption algorithms.
- The solution's viability is demonstrated by implementing it using the P4 programming language in production-grade hardware *i.e.*, an Intel Tofino switch, clearing doubts about its feasibility in real network equipment. We also make our source code publicly available[1].
- Experiments are conducted using traffic traces from 3 different encrypted traffic data sets with different classification tasks. Results shed light on how the proposed solution achieves high classification accuracies in the range of $87.2\% - 95.3\%$ while consuming on average only between $8.3\% - 9.8\%$ of total hardware resources available on the switch.

[1]https://github.com/nds-group/ETC_NOMS_2024.

The rest of the paper is structured as follows. In Section II, the state-of-the-art and related work in the areas of ETC and in-switch inference are presented. Next, Section III provides details on the ETC framework, the offline modelling phase, and the specifics of the in-switch deployment. In Section IV, the details on the testbed, use cases and experiments are provided, with the ensuing results presented and discussed in Section V. The paper is then concluded in Section VI.

II. State of the Art

The proliferation of encryption techniques in today's data networks has fostered an interest in ETC. In this section, related work is examined in view of establishing the state-of-the-art in terms of (i) encrypted traffic classification, and (ii) in-switch inference, thus, motivating in-switch ETC.

A. Encrypted traffic classification (ETC)

As network traffic encryption becomes more present in today's networks, techniques for ETC have evolved from port-based techniques [25]–[27] to payload-based techniques based on DPI [28]–[30], and now to ML and DL techniques [2]–[10].

Port-based ETC. Many applications and services use well-known port numbers assigned and managed by the Internet Assigned Numbers Authority (IANA) [31]. As such, it is often relatively easy to identify traffic from different applications by examining the port numbers [25]. Other features like packet sizes have also been associated with port numbers to enhance traffic classification [26]. These port-based ETC techniques are easy to deploy but they are not effective in the face of port randomization techniques or masquerading, and are vulnerable to port deception attacks [32].

DPI-based ETC. With growing security challenges, encryption techniques have greatly evolved and ETC is now past the port number era [30]. Previous works have proposed to examine the packet payload with DPI middleboxes and derive patterns for traffic classification [28]. These techniques have proven to be more robust than the port-number-based ones and they yield high accuracies and low false positive rates. However, as most encryption techniques randomize bit streams of payloads and anonymize their content, ETC becomes very challenging to DPI algorithms [30]. Some works propose to first decrypt the payloads and then apply DPI [28], but this could compromise data privacy. Blindbox [33] proposes a new protocol and encryption schemes that enable examination of the payload without decrypting it. However, the scheme is evaluated with HTTPS and the performance with other encryption schemes is uncertain.

ML and DL-based ETC. With the difficulties DPI faces to analyze encrypted traffic, ML and DL techniques have taken the stage as the state-of-the-art for ETC. Early works like Li *et al.* [29] propose to combine DPI with ML to enhance the ETC process and benefit from both kinds of techniques. Others solely employ ML and DL, exploiting features computed from packet headers and statistics derived from the packet sizes and packet arrival times [4]–[10]. While these solutions have been shown to achieve high classification accuracy in the tasks they handle, they are mainly run offline in servers outside the network and do not enable real-time ETC.

ETC in SDNs. With the adoption of the SDN architecture with programmable control planes, many efforts have been made to embed ML/DL models into the control plane for traffic classification [11]. In some cases, the models do not run in real-time [15]–[17] and are only different from the offline techniques in that the models run in the SDN control plane. Other solutions propose to enable ETC on-the-fly by making the models to adapt to changes in network traffic or to classify live traffic on-the-go [12]–[14]. Despite their ability to enable real-time ETC, the involvement of the control plane in the ETC process induces communication latencies in the order of tens to hundreds of milliseconds [34], which exceed the strict latency requirements of modern applications and environments like the metaverse [35].

Data plane-assisted ETC. In a bid to reduce the load on the control plane and reduce some of the latency involved with control plane ETC, some works propose to offload some ETC-related tasks to the data plane. Sanvito *et al.* [36] propose to alleviate a DPI engine of some of its tasks by delegating some computation tasks to stateful SDN data planes. This reduces the required bandwidth between switches and the DPI engine as well as the computation power required by the DPI. However, the solution is not fully based in the data plane and still requires DPI middleboxes. Other works propose to perform ETC in the data plane, but in SDN home gateways [13], [14]. Yet, these solutions are specific to home gateways which are CPU-based and would not enable the line rate inference that would be possible with programmable switch Application-Specific Integrated Circuits (ASIC). Ultimately, this leaves room for improvement in terms of porting ETC to the data plane in order to enable high-throughput and low-latency inference at line rate and in real time.

B. ML inference in programmable switches

As P4-programmable switch ASICs like Intel Tofino [19] become more available, a strong interest in running ML models in the data plane has been generated. However, programmable switches are highly constrained environments with a number of strict constraints. (i) *Low available memory*: switches typically have only a few hundred kilobytes of Ternary Content-Addressable Memory (TCAM), and a few tens of megabytes of Static Random Access Memory (SRAM). Most other resources are also highly limited. (ii) *Limited support for mathematical operations*: switch ASICs are not designed to be complex calculators. They only support basic operations like additions, subtractions, bit-wise operations, and logical operations. There is no support for floating-point operations. (iii) *Limited number of operations per packet*: to maintain line rate packet processing, most operations *e.g.*, access to stateful registers can only be performed once per packet, and no loops are supported.

As such, any data plane application that will run in the switch must take into account all the constraints above. This makes training of models in the data plane completely out of

question and places a limit on the nature and complexity of ML tasks that can be delegated to the switch, as well as on the kind of models that can be successfully mapped into the switch. In in-switch inference, ML models are typically trained offline in servers, or in the control and application planes of SDN networks. Upon training, the models are then mapped into the switch Match and Action (M/A) pipeline.

Due to their simple logical structure and limited operations involved at inference, DT and RF models are the most successful in terms of in-switch deployment for traffic classification [20]–[24], albeit with tests on unencrypted traffic. These solutions can be broadly categorized based on the nature of the classification target; individual packets or flows, which are groups of packets with the same 5-tuple (source IP, destination IP, source port, destination port, and transport protocol).

In the packet-level solutions [20]–[23], the features used for ML inference are mainly header fields extracted from single packets such as packet sizes, header lengths, window sizes, and transport protocol flags. While these features are sufficient to tackle some classification problems, they fall short when tackling challenging tasks or encrypted traffic, since some of the header fields could be encrypted or randomized by encrypted algorithms [30]. As such, packet-level inference falls short of being a good solution for in-switch ETC.

Flow-level solutions like Flowrest [24], INC [37] and Friday *et al.* [38] employ additional flow-level features like minimum, maximum and average packet sizes, minimum, maximum and average IATs, the current packet length, and the duration of the flow. These features are used in addition to the packet-level header information which are the sole features in packet-level classification. Nonetheless, none of the above works specifically targets ETC. Moreso, INC [37] and Friday *et al.* [38] only focus on malware detection tasks and are not directly extensible to multi-class ETC problems. A few works exploit the data plane to facilitate the ETC process by offloading some of the inference tasks like feature extraction and initial traffic filtering to the data plane [36], [39]–[43]. However, none of them fully encodes a ML model into the switch data plane for in-switch inference.

Since most of the flow-level statistics like those based on the packet size and on the packet arrival times are not affected by encryption algorithms, they have been exploited for ETC in non-in-switch solutions [4], [15]. In addition, previous work has demonstrated that a flow can be successfully classified after observing only the first few packets of the flow [44], and this concept is applied in most existing flow-level solutions like Flowrest [24]. Also, Lv *et al.* [5] demonstrate in an offline setting that a secondary voting RF model performs well on ETC tasks, making RF models a good pick for both in-switch ML inference and ETC. Building on these state-of-the-art ideas, this paper proposes a fully in-switch ETC solution that applies an RF model for line rate inference on flows in programmable switches, after observing only the first few packets, and using only features based on the packet size and the packet arrival times, which yield high accuracy and are generally unaffected by encryption algorithms [4], [45].

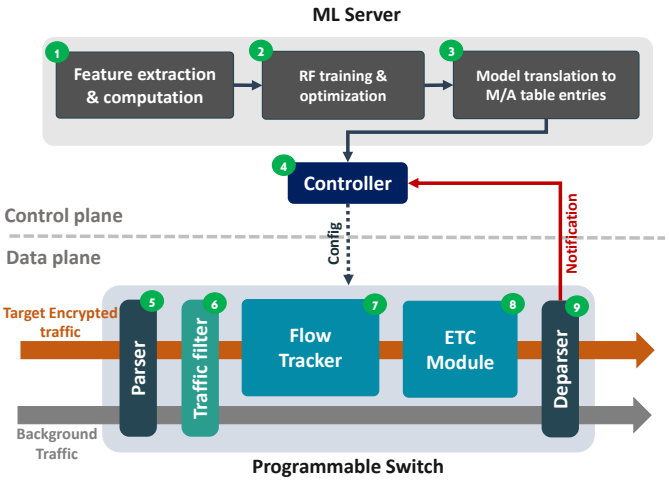


Fig. 1: Overview of the proposed in-switch ETC framework.

III. In-switch Encrypted Traffic Classification

The ETC solution is designed as a fully in-switch model for high-speed inference in the data plane. Figure 1 presents an overview of the framework which is detailed out next. The in-switch ETC framework is designed from the traditional SDN architecture, portraying both the control plane and data plane components as shown in Figure 1. The core network is depicted as an SDN with programmable control and data planes, with the data plane comprising P4-programmable switches. Traffic flows through these switches before arriving the desired application servers. The goal of the ETC solution is to classify this traffic as it flows through the network devices, thereby achieving real time and line rate inference with high throughput and low latency. The ETC framework comprises two main components; (i) the offline/control plane component, where the offline RF model preparation is done, with its elements numbered ①-④ in Figure 1, and (ii) the data plane component, where in-switch ETC happens, with its parts numbered ⑤-⑨. The details of these parts are described next.

A. Offline RF model preparation

Due to the constraints of programmable switches described in Section II-B, the entire feature extraction, selection and RF model training phases take place offline in a server.

Feature extraction and selection. In Step ①, two main features are extracted from the raw pcap files using Tshark [46] and Scapy [47]. These are the packet size, and the packet arrival time. These two features are generally unaffected by encryption algorithms and so would be available for ETC most of the time. Port numbers and other header fields that might be affected by dynamic port number schemes and masquerading are excluded. Flow statistics are then computed in Python and the following nine features are considered for in-switch inference; *the maximum, the minimum, the total* and *the mean* of the two base features; *packet sizes* and *IATs*, and *the current packet size.* The total IAT is the duration of the flow. In the offline ML phase, additional statistics like standard deviations, quartiles, skews and kurtosis of the

different feature distributions are employed but these are not considered in the model analysis for in-switch inference since they are difficult to compute in the switch. They are used to verify if better model performances can be obtained offline with more complex features, hence shedding light on the limits of the in-switch model if any.

RF training and optimization. With all the features computed, Step ② is then invoked. The entire process is summarized in Algorithm 1. It starts with a grid search for RF model hyper-parameters notably the maximum depth of trees, and the number of trees. Once the best hyperparameters are chosen, an initial RF model, $initial\_model$ is trained using all the features. From the feature importance attribute of the trained RF model, the features are ranked based on their importance expressed by their mean decrease in impurity (MDI). Then another model training step is performed, with the features added one by one in order of importance until a high accuracy beyond a set threshold acc_{th} is attained. This enables the selection of the smallest number of features, $selected\_features$, that is required to achieve high accuracy. Once the final feature set and hyperparameters are chosen, the final model is trained and evaluated through a 12-fold cross-validation. The model training is done using the open source Scikit-Learn library [48]. An important aspect of the modelling process is that the constraints of the switch are taken into account. A specific example is in the case of TCAM, where there is a maximum size N in bits that a ternary match key can have. To ensure that all codes mapping paths to tree leaves generated in Step ③ meet this requirement, just like in Flowrest [24], the `max_leaf_nodes` parameter of the `RandomForestClassifier()` object is used.

Algorithm 1: RF training and optimization

```
   Data: features, data, labels
   Result: selected_features, best_hyperparamaters, final_model
 1 best_hyperparameters = grid_search_rf(data, labels, features);
 2 initial_model = train_rf_model(data, labels, features,
     best_hyperparameters);
 3 sorted_features = rank_features_by_importance(initial_model);
 4 selected_features = [ ];
 5 accuracy_threshold = acc_th;
 6 for feature in sorted_features do
 7     selected_features.append(feature);
 8     model = train_rf_model(data, labels, selected_features,
         best_hyperparameters);
 9     accuracy = cross_validate(model, data, labels);
10     if accuracy > acc_th then
11         return selected_features;
12     else
13         continue;
14 final_model = train_rf_model(data, labels, selected_features,
     best_hyperparameters);
15 final_accuracy = cross_validate(final_model, data, labels);
16 output_results(selected_features, best_hyperparameters, final_model);
```

RF model mapping to switch M/A tables. Once the final trained RF is obtained, it then has to be mapped unto the Protocol Independent Switch Architecture (PISA) pipeline of the switch. Many different schemes for mapping RFs into the switch pipelines have been proposed such as Planter [20], pForest [49], and Mousika [22]. Amongst these schemes, Planter [20] is the state-of-the-art scheme adopted by most other recent works like Flowrest [24] and Henna [21], and in modified versions by INC [37] and Friday *et al.* [38]. As such, the Planter [20] scheme is also adopted by this work.

In the adopted mapping scheme, the model is split into two sets of tables; *feature tables*, and *code tables*. For each feature, all the thresholds at tree nodes where this feature is used are encoded in a single table across all trees of an RF model. The thresholds are then organized into ranges, with each range between two thresholds assigned a code word. Combinations of the code words from all the feature tables are encoded in a code table per tree, which represents the paths to tree leaves and hence determines the class of a sample. More details about the mapping scheme can be found in the original paper [20]. Once the models are converted into M/A table entries, they are passed to the controller in Step ④ which communicates with the switch ASIC via the Barefoot Runtime Interface (BRI) to inject configurations or receive packet digest information.

B. In-switch ETC design

The in-switch component of the proposed ETC framework is based on Flowrest [24] and has five main blocks numbered ⑤-⑨ in Figure 1. The blocks are described next.

Parser. Upon entering the switch, packets are parsed in Step ⑤ in the Ingress Parser and their headers are extracted and stored in metadata in the Packet Header Vector (PHV) which runs across the switch pipeline. From the intrinsic metadata, the packet arrival time is recorded, while the packet size is extracted from the IP header. Other information like the transport protocol, source and destination IP and port numbers are also extracted to form the 5-tuple that identifies the packet. From the parser, the packet moves into the Ingress Control, where the actual processing and inference happens.

Traffic filter. This block in Step ⑥ is made up of a M/A table which has information on flows and whether they are in the target domain for classification and if so, whether they have already been classified or not. If the flow 5-tuple extracted in the parser is matched on the traffic filter table as not belonging to the target traffic domain, it is forwarded without any action or dropped if it is just background traffic. If it is identified as already classified, then it is forwarded accordingly without being classified again. In the case where the packet belongs to a flow that has not been classified and has to be classified, it proceeds to the flow tracking block.

Flow tracker. The flow tracking block in Step ⑦ has two main phases. The first is flow identification and tracking, during which CRC32 (flow identifier) and CRC16 (register index) hashes of the 5-tuple of header information are computed and used to identify flows and retrieve their associated data from stateful registers in the SRAM of the switch. When the hashes are computed, a check is done to ensure that the flow information retrieved from the registers actually corresponds to the current packet. This is done to avoid hash collisions. In the second phase, flow-level features are computed using Intel Tofino's `RegisterAction` extern to read and update per-

flow features. These features are then passed on to the ETC module for inference on flows with the encoded RF model.

ETC module. Once the first n packets of a flow are received[2], the ETC inference module is activated. This module leverages the encoded model for inference by first using range matches to compare flow-level features with feature tables and compose unique codewords that identify a specific path to a leaf on a tree. The generated codewords are then compared with ternary matches against codeword tables, each corresponding to a tree. The feature tables are applied simultaneously while the code tables are also applied simultaneously in the next stage. When all tree decisions are made, a voting table is employed to produce the final classification as the majority vote of the different tree outputs. In the case where all trees produce a different result, the most accurate individual tree result is picked. A register is used to temporarily store the classification result of the flow. To achieve this, the n-th packet which triggered the classification is recirculated and used to update the register with the classification result so that subsequent packets of the flow that arrive before the filter table is updated are no longer classified but forwarded early after the register is checked and the class information is retrieved.

Deparser. When a classification result has been reached, a packet digest is packed with the 5-tuple of the flow, the packet count, the classification result, and the register index where that flow's information is stored. The digest is sent to the controller which upon receiving it sends an update to the traffic filter table, adding the class of the flow to the corresponding flow ID 5-tuple so that subsequent packets of the flow are no longer classified but are forwarded directly when they go through the traffic filter. The classification results are also saved in a CSV file for statistical purposes. It is important to remark that this communication with the controller is completely off the critical path of the classification process and does not introduce any latency to the ETC process which happens entirely in the switch.

IV. Experimental Setup

The prototype of the in-switch ETC solution is implemented in a production-grade Intel Tofino switch using the P4 language. The programs have between $663 - 720$ lines of P4 code, with in-switch RF models of maximum depth in the range $16 - 20$, number of trees in the range $1 - 5$, and $4 - 5$ features retained at the end of the feature selection process. The solution is then evaluated on an experimental testbed with switches and servers, with tests on 3 use cases derived from 3 ETC datasets. Details of the testbed, use cases, datasets, baseline and evaluation metrics are presented next.

A. Hardware setup

Three Edgecore Wedge100BF-QS programmable switches with an Intel Tofino BFN-T10-032Q chipset and 32 100GbE QSFP28 ports are used to build a testbed to validate the ETC solution. The testbed also comprises two servers with Intel 8-core Xeon processors running at 2GHz, with 48GB of RAM, and QSFP28 interfaces. The servers and switches are connected via 100 Gbps optical links. The switches run Intel's Software Development Environment (SDE) version 9.7.0 and the Open Network Linux (ONL) operating system. The SDE also comes with the Intel P4 Insight tool [50] which provides a graphical user interface with a detailed view of the mapping of P4 programs onto Tofino switch hardware resources. This tool enables a thorough assessment of the switch resource usage by the ETC framework. A Python controller is written that interacts with the switch via the Barefoot Runtime Interface (BRI) for the initial configuration of the switch, such as setting up ports and loading the trained and translated RF model as M/A table entries.

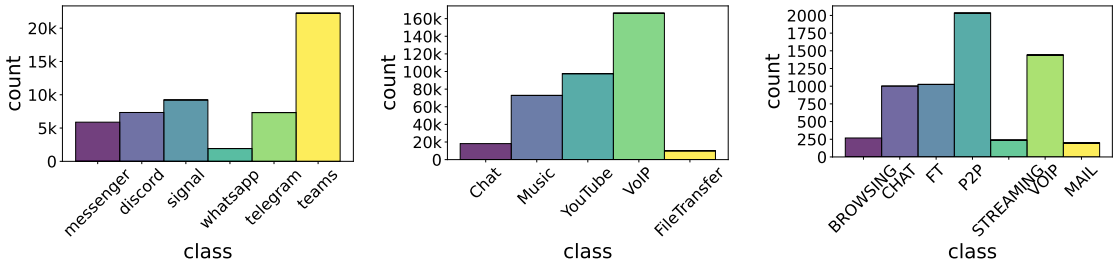
The MoonGen [51] traffic generator is exploited to generate and inject background traffic at 40 Gbps into the switch, while the target encrypted traffic data is simultaneously replayed as pcap files using Tcpreplay [52]. The background traffic is included in the experiments in order to have the switch operate at all times on high-throughput traffic and increase the realism of the evaluation, however, as shown in Figure 1, only the target encrypted traffic is classified by the in-switch RF model. Therefore, all results in Section V refer to the accuracy of the solution in recognizing the different classes of encrypted traffic only, and are not affected by the amount or nature of the background traffic.

B. Use cases and datasets

The targeted use cases include; encrypted instant messaging application fingerprinting, QUIC traffic classification, and VPN traffic classification.

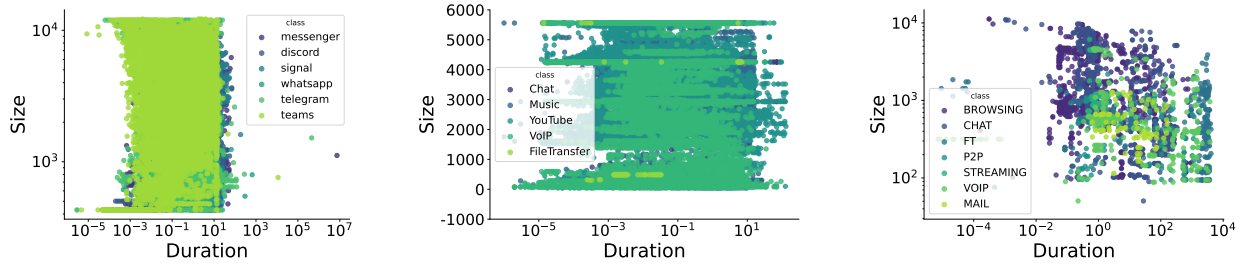
Encrypted instant messaging application fingerprinting. With the rapid proliferation of mobile instant messaging applications (IMAs) with end-to-end encryption, identifying traffic from such apps is relevant for network management tasks like bandwidth allocation. An application fingerprinting use case is built on the Encrypted Mobile Instant Messaging Traffic Dataset from NIMS Lab (NIMS IMA) [53], [54]. It is divided into 7 files in the zip format. 6 of these files contain traffic from commonly used Instant Messaging applications; *Discord*, Facebook *Messenger*, *Signal*, Microsoft *Teams*, *Telegram* and *WhatsApp*. The 7^{th} file contains encrypted traffic that is not from any of the IMAs, divided into 4 classes. The dataset contains $4,062,861$ packets from $92,361$ flows from these 10 different classes. In this work, only the problem of distinguishing between the 6 IMAs is considered (as in [53]). Figure 2a shows the number of samples per class and clearly shows that the data is not well balanced, with much more flows for the *Teams* application and the lowest number of flows for *WhatsApp*. Figure 3a shows a projection of the dataset on 2 axes with logarithmic scales for two features: the total duration of flows, and the sum of the size of all the packets in the flow. Apart from a few outliers, this figure shows no obvious separation of the data across classes, implying that the classification problem is not trivial.

[2] n is the number of flow packets that provide enough information for computing and using flow-level features. $n = 4$ and $n = 8$ were the best values in the experiments considered.



(a) (b) (c)

Fig. 2: Distribution of flows across classes in the datasets namely, (a) NIMS IMA, (b) Netflow QUIC, and (c) ISCX VPN.



(a) (b) (c)

Fig. 3: 2D projection of the flow duration and total packet size features in the different datasets namely, (a) NIMS IMA considering the first 8 packets of each flow, (b) Netflow QUIC considering the first 4 packets of each flow and (c) ISCX VPN considering the first 8 packets of each flow.

QUIC traffic classification. QUIC is an encrypted protocol initially proposed by Google which operates at the transport layer and is connection-oriented although based on UDP. It is widely used today and a number of works have targeted QUIC traffic classification [4], [55]. A QUIC classification task is designed with the Netflow QUIC dataset from [55]. It is a labeled dataset with 5 different classes: *Google Hangout Chat*, *Google Hangout VoIP*, *File Transfer*, *Youtube*, and *Google Play Music*. This dataset is significantly larger than the NIMS one with $365,000$ flows and a total of 136 millions packets. The distribution of flows across the 5 classes is not well-balanced as shown in Figure 2b. Figure 3b shows a projection of the dataset across 2 axes showing the total duration of the flows and the sum of the size of the packets in each flow. It portrays how the classes are separated, for example only the *YouTube* class seems to have duration between 10 and 100s for a total size around $3,000$ bytes. This makes these features important for the classification task.

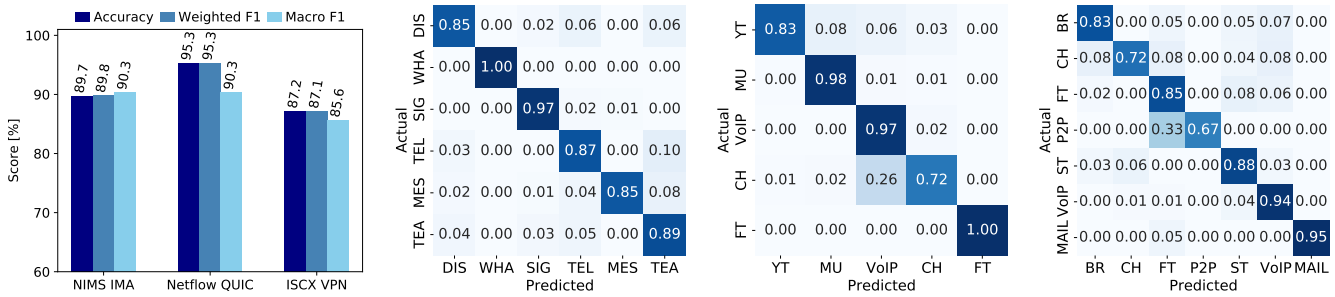
VPN traffic classification. Virtual Private Network (VPN) tunnels are a popular tool for achieving data privacy and security in today's encrypted data networks. Thus, classifying VPN traffic is a growing area of interest. A VPN traffic classification use case is designed around the ISCX-VPN-NonVPN-2016 Dataset [8]. This is a popular labeled dataset made available by the Canadian Institute of Cybersecurity (CIC) at the University of New Brunswick (UNB). It comprises about 28GB of traffic data captured using tcpdump and Wireshark. A subset of this dataset is made of VPN data, generated using an external VPN service provided connected to the testbed using OpenVPN in UDP mode. This subset of the dataset includes 7 classes of encrypted traffic: *Browsing*, *Email*, *Chat*, *Streaming*, *File Transfer*, *VoIP*, and *P2P*. For this work the dataset was preprocessed from the raw PCAP files to keep only the VPN subset (ISCX VPN) and the packets were aggregated into flows. To this end, all packets matching the same 5-tuple were considered to be part of the same flow. This resulted in $4,960$ flows whose distribution into the different classes in the dataset is shown in Figure 2c.

Extracted features. For each of the datasets and for each flow, 2 base features were considered: the IAT and the packet length. From these 2 base features, the datasets were enriched with additional statistical features that were computed from the packet information: the minimum, the maximum, the sum, the mean, the median, standard deviation, first and last quantile, the skew and the kurtosis. As described in Section III-A, all these features were used to train the offline baseline RF and XGBoost models, while only a smaller subset of features was used for the online RF models deployed in the switch.

C. Baseline and metrics

The ETC framework is compared to an offline baseline which uses fully-grown RF and XGBoost models, with as many as 21 features, some of which are not easy to compute in the switch, *e.g.*, skews, kurtosis, *etc.* This model is not subject to the constraints of programmable switches and enables an assessment of by how much classification performances could be degraded upon moving to in-switch operation.

The performance of the ETC solution and the baseline are assessed via classical metrics notably; (i) *accuracy*, and



(a) (b) (c) (d)

Fig. 4: Classification performance from in-switch ETC experiments; (a) bar plot showing accuracy, macro and weighted F1 scores, and confusion matrices for (b) NIMS IMA, where DIS = Discord, WHA = Whatsapp, SIG = Signal, TEL = Telegram, MES = Messenger, TEA = Teams; (c) Netflow QUIC, where YT = Youtube, MU = Google Play Music, VoIP=Google Hangout VoIP, CH = Google Hangout Chat, FT = File Transfer; (d) ISCX VPN, where BR = Browsing, ST = Streaming.

(*ii*) *F1-score*. These metrics are calculated from four key measures of a classification problem, *i.e.*, true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN), as follows.

- **Accuracy** captures the amount of samples that are predicted correctly, as (TP + TN)/(TP + FP + TN + FN).
- **F1-score** is the harmonic mean of recall and precision and is widely used to compare model performances. It is computed as 2TP/(2TP + FP + FN).

For the F1-Score, the final value is averaged over all classes without weights in the case of the macro average, or weighted by the number of samples of each class in the dataset, in the case of the weighted average. Individual class performances are also presented in confusion matrices.

V. Results and Discussion

The ETC framework is evaluated in terms of classification accuracy, resource usage, and packet processing latency.

A. Classification accuracy

The classification performance of the in-switch ETC solution over the three use cases is presented in Figure 4. Overall the ETC models achieve high classification accuracy in the tasks targeted as shown in Figure 4a, starting from 87.16% for the ISCX VPN dataset, which seems to be the most challenging, through 89.74% for the NIMS IMA dataset, to 95.28% for the Netflow QUIC dataset, where the highest classification accuracy is obtained. In terms of the macro F1 score which gives a measure of how good the classifier is irrespective of the number of samples of each class, the scores are also high, ranging from 85.64% in ISCX VPN through 90.29% in NIMS IMA to 90.31% in Netflow QUIC. This shows that although the datasets are generally unbalanced between classes as described in Section IV-B, the ETC models are still able to do well in distinguishing them. Factoring the imbalance into account and considering the weighted F1 scores, the scores are higher, ranging from 87.14% in ISCX VPN through 89.82% in NIMS IMA to 95.29% in Netflow QUIC. The higher weighted F1 scores mean that the models tend to perform very well on the most represented classes and so pull the overall score upwards when the average is weighted by number of samples.

The confusion matrices in Figures 4b, 4c, and 4d show the classification performance per class and per use case of the scenarios considered. For IMA fingerprinting (NIMS IMA) shown in Figure 4b, the ETC model excels at identifying *WhatsApp*, and *Signal* with 100% and 97% respectively. For all the other IMAs, the ETC model never goes below 85% accuracy. In the case of QUIC traffic classification portrayed in Figure 4c, for the *File Transfer*, *Google Play Music*, and *Google Hangout VoIP* classes, the ETC model achieves over 97% accuracy, coming short only in the *Youtube*, and *Google Hangout Chat* classes where it achieves 83% and 72% respectively. This lower score for the *Google Hangout Chat* class could be attributed to the diverse nature of its traffic, and to the reduced amount of data used for the in-switch tests. The VPN traffic classification (ISCX VPN) case shown in Figure 4d is the most challenging. Just like in the QUIC classification use case, *Chat* traffic remains one of the hardest to identify with 72% of samples correctly predicted, seconding only *P2P* with just 67% accuracy. In all cases *P2P* gets confused for the *FT* class. For the rest of the classes, the performance of the ETC solution is quite good, ranging from $83\% - 95\%$.

Juxtaposing the classification performance of the in-switch ETC solution with those of the offline baseline Python RF and XGBoost models, it turns out that the in-switch models are mostly on par with the offline baseline models as shown in Table I. The baseline XGBoost result shows no significant difference with that of baseline RF, confirming that the choice of RF models ensures high performance comparable to other model types. The in-switch model performs poorer in a few cases like in the *Google Hangout Chat* class of the Netflow QUIC dataset, where it achieves only 70% accuracy, while the offline baseline models achieve up to 96%. This also happens in some classes of the ISCX VPN dataset, where the offline baseline also performs better. This lower performance could be attributed to the approximations that are made to features to fit the switch hardware constraints. Nonetheless,

Dataset	Classes	Offline baseline (Python)		Online (P4) [%]
		RF [%]	XGBoost [%]	
NIMS IMA	Discord	89.22	90.48	84.21
	Messenger	89.38	92.32	91.23
	Signal	99.1	99.63	96.40
	Teams	93.55	94.58	87.39
	Telegram	88.89	90.22	82.54
	WhatsApp	94.08	97.23	100
Netflow QUIC	YouTube	85.42	85.48	87.43
	Music	95.05	94.83	96.34
	VoIP	93.92	93.70	97.01
	Chat	96.62	96.46	70.82
	File Transfer	99.93	99.92	99.92
ISCX-VPN	Browsing	95.33	94.94	86.42
	Chat	90.72	91.88	78.26
	FileTransfer	96.70	98.07	85.71
	P2P	91.77	93.25	80.00
	Streaming	94.56	95.51	80.00
	VoIP	96.50	96.92	91.46
	Mail	95.10	96.41	97.67

TABLE I: Summary of the performance of the offline baseline models and the online ETC models in terms of F1-score.

in some classes like *Google Play Music*, *Google Hangout VoIP* and *Youtube*, the in-switch ETC model outperforms the offline baseline. This is attributable to the efficient feature and model selection process used in the in-switch RF modelling that enables attaining high accuracy with as few features as possible. Ultimately, these results lead to the conclusion that ETC can be carried out in real-time and at line rate in switches with accuracies on par with those of offline methods.

B. Resource usage

The main role of the switch is not to run ML models but to forward packets. As such, it is important to assess the memory footprint of the in-switch ML-based ETC. Using the P4 Insight tool described in Section IV-A, the resource consumption of each P4 program is analyzed with the details presented in Table II. Overall, the model deployments consume less than 10% on average of the total available resources on the switch. This leaves room for cohabitation with other switch functions.

Considering individual resources, the models only consume between $4.0\% - 5.5\%$ of SRAM, which is used for the stateful registers that store flow information. This leaves a lot of SRAM for other switch functions. The Very Long Instruction Words (VLIW) which are used for mathematical operations, are also only consumed at about $5.5\% - 6.3\%$, allowing many other operations to potentially run simultaneously.

TCAM is the most critical switch resource. It is used for the ternary and range matches employed in the M/A tables of the RF models, alongside its corresponding input crossbar (Xbar) that carries the match keys. TCAM consumption varies across use cases. In the Netflow QUIC dataset, as the RF has only 1 tree, less TCAM is required, and fewer bits of Xbar are needed, explaining the lower values in Table II. The ISCX VPN dataset has trees with more leaves which require more ternary match bits in its tree M/A tables. That explains the 24.2% of Xbar required for its match keys. Regarding the action data bus bytes that carry action data, consumption is as low as 7% in Netflow QUIC but rises to 12.4% in NIMS and 13.9% in the ISCX VPN case due to trees having more leaves hence, longer code words. The P4 programs use $10 - 12$ M/A stages out of the 12 available. This does not leave any stages for post-inference processing in the ingress pipeline. However, it is possible to do additional processing in the egress pipeline where all the 12 stages are again accessible.

Resource	Datasets		
	NIMS [%]	Netflow QUIC [%]	ISCX VPN [%]
Action Data Bus Bytes	12.4	7	13.9
Logical Table ID	16.1	15.6	15.6
SRAM	4.4	5.4	4.4
TCAM	11.5	7.3	15.3
Ternary Match Input Xbar	18.9	11.1	24.2
VLIW Instruction	6.3	5.5	6.3
Total Avg. Resource Usage	9.09	8.33	9.76
Number of M/A Stages	12	10	12

TABLE II: Summary of the usage of key switch resources.

To put all these numbers into perspective, it is worth noting that deploying the largest ETC model alongside the baseline P4 program for core L2/L3 switching functions, *i.e.*, `switch.p4` increases overall resource consumption by only 25% of what the baseline requires. This sheds light on the memory efficiency of the ETC models.

C. Latency

The main motivations for in-switch ETC are low latency and high throughput. P4 programs compiled and deployed in the switch run at line rate. Thus, all the ETC models run at line rate. P4 Insight provides details on latency in terms of the number of clock cycles in both ingress and egress processing. Using the number of cycles and the clock frequency of the Tofino switch ASIC, the average packet processing latency of the programs in nanoseconds is computed. Across the 3 in-switch ETC use cases, the latency ranges from $383 - 398$ nanoseconds, confirming a sub-microsecond delay for in-switch ETC, many orders of magnitude more than the milliseconds of control plane solutions.

VI. Conclusion

An ETC solution is proposed that exploits recent advances in data plane ML inference to accelerate ETC by running inference on flows at line rate, fully in programmable switches. The solution is implemented in production-grade Intel Tofino hardware and tested with public traffic traces, revealing how it achieves a classification accuracy on par with unconstrained offline baselines, despite the constraints or hardware environments. This performance is obtained while maintaining sub-microsecond delay and consuming less than 10% of hardware resources. This sets a new state-of-the-art for high-speed ETC in SDN data planes. Future work could explore improving the classification accuracy, experimenting with datasets featuring a much larger number of classes, and tackling use cases where traffic is entirely tunneled such that it is difficult to identify flows via their 5-tuples, in which case classifying entire sessions might be more appropriate.

Acknowledgments

This work has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under grant agreement no. 101139270 "ORIGAMI" and the Marie Skłodowska-Curie grant agreement no. 860239 "BANYAN".

REFERENCES

[1] Google, "HTTPS encryption on the web – Google Transparency Report," 2023.

[2] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, vol. 25, 2015.

[3] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Communications Magazine*, vol. 57, 2019.

[4] I. Akbari, M. A. Salahuddin, L. Ven, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, and S. Tuffin, "Traffic classification in an increasingly encrypted web," *Communications of the ACM*, vol. 65, 2022.

[5] G. Lv, R. Yang, Y. Wang, and Z. Tang, "Network encrypted traffic classification based on secondary voting enhanced random forest," in *IEEE CCET*, 2020.

[6] V. A. Muliukha, L. U. Laboshin, A. A. Lukashin, and N. V. Nashivochnikov, "Analysis and classification of encrypted network traffic using machine learning," in *IEEE SCM*, 2020.

[7] S. Rezaei, B. Kroencke, and X. Liu, "Large-scale mobile app identification using deep learning," *IEEE Access*, vol. 8, 2020.

[8] G. Draper-Gil, A. H. Lashkari, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of encrypted and VPN traffic using time-related features," in *ICISSP*, 2016.

[9] P.-O. Brissaud, J. Francçis, I. Chrisment, T. Cholez, and O. Bettan, "Transparent and service-agnostic monitoring of encrypted web traffic," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 3, 2019.

[10] W. M. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A multi-level framework to identify HTTPS services," in *IEEE/IFIP NOMS*, 2016.

[11] J. Yan and J. Yuan, "A survey of traffic classification in software defined networks," in *HotICN*, 2019.

[12] Z. A. Qazi, J. Lee, T. Jin, G. Bellala, M. Arndt, and G. Noubir, "Application-awareness in SDN," in *SIGCOMM*, 2013.

[13] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in SDN home gateway," *IEEE Access*, vol. 6, 2018.

[14] X. Chen, J. Yu, F. Ye, and P. Wang, "A hierarchical approach to encrypted data packet classification in smart home gateways," in *DASC/PiCom/DataCom/CyberSciTech*, 2018.

[15] A. S. D. Silva, C. C. Machado, R. V. Bisol, L. Z. Granville, and A. Schaeffer-Filho, "Identification and selection of flow features for accurate traffic classification in SDN," in *IEEE NCA*, 2016.

[16] P. Amaral, J. Dinis, P. Pinto, L. Bernardo, J. Tavares, and H. S. Mamede, "Machine learning in software defined networks: Data collection and traffic classification," in *IEEE ICNP*, 2016.

[17] M. R. Parsaei, M. J. Sobouti, S. R. khayami, and R. Javidan, "Network traffic classification using machine learning techniques over software defined networks," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, 2017.

[18] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, 2014.

[19] Intel, "Tofino Switch," Intel Corporation, Mountain View, CA, United States. [Online]. Available: https://tinyurl.com/ycx79k4z

[20] C. Zheng and N. Zilberman, "Planter: Seeding trees within switches," in *SIGCOMM '21*. NY, USA: ACM, 2021.

[21] A. T.-J. Akem, B. Bütün, M. Gucciardo, and M. Fiore, "Henna: Hierarchical machine learning inference in programmable switches," in *NativeNi '22*. NY, USA: ACM, 2022.

[22] G. Xie, Q. Li, Y. Dong, G. Duan, Y. Jiang, and J. Duan, "Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation," in *IEEE INFOCOM*, 2022.

[23] B. Bütün, A. T.-J. Akem, M. Gucciardo, and M. Fiore, "Fast detection of cyberattacks on the metaverse through user-plane inference," *IEEE MetaCom*, 2023.

[24] A. T.-J. Akem, M. Gucciardo, and M. Fiore, "Flowrest: Practical flow-level inference in programmable switches with random forests," in *IEEE INFOCOM*, 2023.

[25] P. Schneider, "TCP / IP traffic classification based on port numbers," 1997. [Online]. Available: https://shorturl.at/htv28

[26] Y.-s. Lim, H.-c. Kim, J. Jeong, C.-k. Kim, T. T. Kwon, and Y. Choi, "Internet traffic classification demystified: On the sources of the discriminative power," in *CoNEXT*. NY, USA: ACM, 2010.

[27] C.-N. Lu, C.-Y. Huang, Y.-D. Lin, and Y.-C. Lai, "Session level flow classification by packet size distribution and session grouping," *Computer Networks*, vol. 56, no. 1, 2012.

[28] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Müller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *IEEE Communications Surveys & Tutorials*, vol. 16, 2014.

[29] Y. Li and J. Li, "Multiclassifier: A combination of DPI and ML for application-layer classification in SDN," in *ICSAI*, 2014.

[30] H. Doroud, A. Alaswad, and F. Dressler, "Encrypted traffic detection: Beyond the port number era," in *IEEE LCN*, 2022.

[31] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire, "Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry," 2011. [Online]. Available: https://shorturl.at/iotE9

[32] F. Hu, S. Zhang, X. Lin, L. Wu, N. Liao, and Y. Song, "Network traffic classification model based on attention mechanism and spatiotemporal features," *EURASIP Journal on Information Security*, vol. 2023, 2023.

[33] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy, "Blindbox: Deep packet inspection over encrypted traffic," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, 2015.

[34] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, and M. Thottan, "Measuring control plane latency in SDN-enabled switches," ser. SOSR '15. NY, USA: ACM, 2015.

[35] J. Xu, K. Papangelis, J. Dunham, J. Goncalves, N. J. LaLone, A. Chamberlain, I. Lykourentzou, F. L. Vinella, and D. I. Schwartz, "Metaverse: The vision for the future," in *CHI EA '22*. NY, USA: ACM, 2022.

[36] D. Sanvito, D. Moro, and A. Capone, "Towards traffic classification offloading to stateful SDN data planes," in *IEEE NetSoft*, 2017.

[37] K. Friday, E. Kfoury, E. Bou-Harb, and J. Crichigno, "INC: In-network classification of botnet propagation at line rate," in *ESORICS*, 2022.

[38] K. Friday, E. Bou-Harb, and J. Crichigno, "A learning methodology for line-rate ransomware mitigation with p4 switches," in *NSS*, 2022.

[39] S. Ogasawara and Y. Takahashi, "Performance analysis of traffic classification in an openflow switch," in *IEEE CIoT*, 2017.

[40] A. Bianco, P. Giaccone, S. Kelki, N. M. Campos, S. Traverso, and T. Zhang, "On-the-fly traffic classification and control with a stateful SDN approach," *IEEE ICC*, 2017.

[41] M. Hayes, B. Ng, A. Pekar, and W. K. Seah, "Scalable architecture for SDN traffic classification," *IEEE Systems Journal*, vol. 12, 2018.

[42] B. Ng, M. Hayes, and W. K. Seah, "Developing a traffic classification platform for enterprise networks with SDN: Experiences & lessons learned," in *IFIP Networking*, 2015.

[43] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications," in *NDSS*, 2021.

[44] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 2, 2006.

[45] J. Piet, D. Nwoji, and V. Paxson, "Ggfast: Automating generation of flexible network traffic classifiers," in *ACM SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2023, p. 850–866. [Online]. Available: https://doi.org/10.1145/3603269.3604840

[46] G. Combs, "Tshark," *Wireshark*, 1998. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html

[47] R. R. S, R. R, M. Moharir, and S. G, "Scapy- a powerful interactive packet manipulation program," in *ICNEWS*, 2018.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.

[49] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pForest: In-network inference with random forests," *CoRR*, vol. abs/1909.05680, 2019.

[50] Intel, "P4 Insight," https://shorturl.at/ceN59.

[51] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "Moongen: A scriptable high-speed packet generator," in *IMC*, 2015.

[52] A. Turner and F. Klassen, "Tcpreplay," 2013.

[53] Z. Erdenebaatar, B. Nandy, N. Seddigh, R. Alshammari, M. Elsayed, and N. Zincir-Heywood, "Instant messaging application encrypted traffic generation system," in *IEEE/IFIP NOMS*, 2023.

[54] Z. Erdenebaatar, "Encrypted Mobile Instant Messaging Traffic Dataset." [Online]. Available: https://shorturl.at/gpw02

[55] V. Tong, H. A. Tran, S. Souihi, and A. Mellouk, "A novel QUIC traffic classifier based on convolutional neural networks," in *IEEE GLOBECOM*, 2018.