# O'MINE: A Novel Collaborative DDoS Detection Mechanism for Programmable Data-Planes

Enkeleda Bardhi*
*Delft University of Technology*
*E.Bardhi-1@tudelft.nl*
*Co-first authors

Chenxing Ji*
*Delft University of Technology*
*C.Ji@tudelft.nl*
*Co-first authors

Ali Imran
*University of Michigan*
*imranali@umich.edu*

Muhammad Shahbaz
*University of Michigan*
*msbaz@umich.edu*

Riccardo Lazzeretti
*Sapienza University of Rome*
*lazzeretti@diag.uniroma1.it*

Mauro Conti
*University of Padua*
*mauro.conti@unipd.it*

Fernando Kuipers
*Delft University of Technology*
*F.A.Kuipers@tudelft.nl*

2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P) | 979-8-3315-9493-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/EuroSP63326.2025.00049

***Abstract*—The emergence of softwarized network devices, like programmable switches and smart NICs, has brought about new and advanced network functionalities. Intelligent decision-making becomes possible at line rate by offloading network functionality from the network control-plane to the programmable data-plane. In this paper, we offload fine-grained Distributed Denial of Service (DDoS) attack detection to the data-plane. The state-of-the-art in this regard, mainly aims to embed Machine Learning (ML) models into the data-plane without compromising on inference accuracy. Besides accuracy, we must consider multiple other factors, like traffic feature availability and false positive rates. To that end, we propose O'MINE: ONE MODEL IS NOT ENOUGH, a novel collaborative detection mechanism comprising lightweight ML models. This maximises the detection accuracy while keeping the false positive rate (FPR) low. We use three state-of-the-art datasets to evaluate the O'MINE algorithm and its ML models. Our results show that O'MINE can detect DDoS attacks with high accuracy (≈98% and ≈96% with full and scarce training data, respectively) and low FPR (≈0.22% and ≈0.72% with full and scarce training data, respectively), outperforming the state-of-the-art. Lastly, O'MINE only consumes a few device resources (≈6% of LUT and ≈4% of FF) on the Xlinx Alevo U250 FPGA we have used for inference at line rate.**

***Index Terms*—DDoS Detection, Programmable Data-Planes, Machine Learning, FPGA**

## 1. Introduction

Software Defined Networking (SDN) [1], which separates the control-plane from the data-plane, has enabled network operators to flexibly program and update their network devices. Mostly, researchers and practitioners have leveraged data-plane programmability to accomplish a wide range of networking-wise tasks, e.g., load balancing [2], [3], security-related mechanisms [4], [5], distributed in-network computing [6], [7], and congestion control [8], [9]. For what concerns security, detecting Distributed Denial of Service (DDoS) attacks is a timeless task in modern networks[1]. In recent years, Machine Learning (ML)-based solutions have been proposed for DDoS detection, for both the control-plane and data-plane [10]–[14]. Deploying ML models in the data-plane is made available by either mapping specific algorithms, e.g., Decision Tree (DT) and Random Forest (RF), as flow tables for programmable switches [5], or engaging high-performing hardware-specific pipelines, e.g., Taurus [13] and Brainwave [15], that enable the deployment of Neural Networks (NNs) and other computation expensive models, with negligible latency [11], [12], [16]–[18].

Despite the benefits of ML and the added computation power in the data-plane, DDoS detection presents some challenges and shortcomings that need to be addressed, listed as follows: *(i)* elevated rate of false positives, meaning that an alarm is raised even though the attack did not happen; *(ii)* lack of attention regarding the data availability in real-world deployments; and *(iii)* need for line rate inference. Modern detection systems share a common bottleneck when it comes to the elevated number of false positive alarms [19]–[22]. State-of-the-art approaches [23], [24] propose post-hoc solutions that operate after the alarms are collected, aiming to address the high false alarm rate, especially in the context of intrusion detection. However, in ML-based DDoS detection, false positives can be mitigated during the design phase. To this end, prior research shows that collaborative detection mechanisms [25], [26] can enhance detection performance while simultaneously reducing the false alarm rate. Building on these findings, we adopt a collaborative detection approach in our work to achieve both improved accuracy and a reduced false alarm rate. Additionally, the design of the detection mechanism should be carefully assessed to ensure a proper assessment of the deployment environment. In particular, the type of the data that is used to train – and consequently needed during inference – of ML-based security mechanisms remains a critical concern that must be carefully addressed [27], [28]. Lastly, the line-rate inference is tightly connected to the assessment of the nature of data. Prior works [11], [29] suggest that extracting different levels, i.e., per-packet or per-flow, of

[1] Akamai Report: https://tinyurl.com/mnv8juu9



DOI 10.1109/EuroSP63326.2025.00049

traffic characteristics to enable DDoS inference in real-time is not always possible.

In view of the above, in this paper, we propose a novel fully collaborative in-network DDoS detection mechanism, named O'MINE: ONE MODEL IS NOT ENOUGH, that distributes the detection process among various programmable network switches that are equipped with lightweight models trained on different sets of data. O'MINE's key feature is leveraging the decision capabilities of multiple models to maximise the accuracy while decreasing the rate of false positives for DDoS detection. Via experimental evaluation, we show that performing collaborative detection provides better results compared to the single model decision on each switch or state-of-the-art mechanisms, with low resources consumed. Additionally, O'MINE pays particular attention to the traffic features used for ML training in terms of the feasibility of extracting them in real-world deployment systems. To this end, we leverage state-of-the-art tools to reason on the feature utility for boosting the model's accuracy, while elaborating on the likeliness of extracting it at line rate speed without sacrificing the latency constraints. Lastly, we evaluate O'MINE in software-based tools for programmable network switches and measure the hardware-wise – using a Field Programmable Gate Array (FPGA) – practicability of adopting our mechanism at line rate without sacrificing the accuracy. In summary, the main contributions presented in this paper are as follows:

- Design and implementation of O'MINE, an end-to-end collaborative DDoS detection mechanism for programmable data-planes that leverages collaborative detection on multiple switches rather than using a single detection model;
- Detailed consideration of features that ML-based DDoS detection mechanisms need for inference in real-world deployments analysing three state-of-the-art network security datasets; and
- Evaluation of O'MINE with both software-based tools and an FPGA-based testbed, comparing it with baseline models and state-of-the-art. O'MINE artifact is publicly available[2]

This paper is organised as follows: in Section 2, we present background information and related work. Subsequently, Section 3 introduces the design of the proposed distributed detection mechanism. After that, in Section 4, we present an extensive evaluation of the proposed mechanism against baseline models and state-of-the-art. Section 5 provides a thorough discussion of the proposed mechanism in terms of challenges and opportunities that are worth investigating for the future. Finally, we conclude in Section 6.

## 2. Background and Related Work

In this section, we provide background in Section 2.1, including information on DDoS attacks in Section 2.1 and on programmable data-planes in Section 2.1. Additionally, in Section 2.2, we first describe the state-of-the-art ML DDoS detection, and later, we elaborate on techniques for enabling learning-based models in network devices.

[2]O'MINE Artifact

### 2.1. Background

**DDoS Attacks:** Distributed Denial of Service (DDoS) is a type of malicious activity on computer networks that originates from multiple distributed sources in the network and which aims to exhaust the target's resources, thereby denying service to legitimate users. The involved parties comprise the attackers, the target resources, and the victims. One can think of the attackers as a set of malware-infected devices (bots) connected to the Internet that collectively execute programmed malicious tasks. Over the years, DDoS attacks have increased both in volume and duration [30]. According to recent reports[3], DDoS attacks now reach rates up to several Tbps. There exist different types of DDoS attacks [31]. The most straightforward DDoS model consists of exhausting the victim's computing resources by opening several connections from multiple sources [32]. Another type of DDoS is the Link Flooding Attack (LFA) that aims at congesting targeted critical network links, leading to the isolation of legitimate users from the network [33]. Given their distributed nature, the mechanism to detect DDoS attacks must be able to adapt to the diverse strategies used by attackers.

**Programmable data-plane:** SDN [34] proposed to separate the control and data-planes, which were traditionally combined in one device (router), thereby making the control-plane programmable. Later, the concept of reconfigurable match-action hardware [35] was introduced to also enable data-plane programmability. The numerous advantages brought by programmable data-plane devices, such as in-band telemetry and customizable processing logic, have sparked a significant amount of research over the past years [36]–[39]. P4 [40] is the predominant domain-specific networking language to program programmable data-plane-enabled hardware, such as FPGAs and smartNICs. Along with P4, the Portable Switch Architecture (PSA) [41] was introduced as a standard packet forwarding processing model. The PSA divides the processing into two parts: ingress and egress. Both include a parser, processing logic, and a deparser. This allows the P4 program to define and parse custom headers and custom processing packet processing logic. In addition, PSA has introduced stateful components, allowing each device to maintain a certain level of states locally.

### 2.2. Related Work

In this section, we first present related work on learning-based DDoS detection mechanisms. Later, we highlight existing techniques for enabling learning-based models in network devices.

**ML-based DDoS Detection:** Several solutions exist for in-network DDoS detection in programmable data-planes [6], [42]–[46]. The main focus has been the detection of specific forms of DDoS attacks, e.g., Amplified Reflection, SYN Flood, and volumetric DDoS. In particular, prior work optimises the integration of detection and defence policies in network devices by leveraging border routers [42], distributed ledger technology [44], switches [43], [45], etc. On the other

[3]Netscout Threat Report: https://www.netscout.com/threatreport/

hand, multiple learning-based DDoS detection mechanisms emerged [11], [12], [29], [47] that are capable of identifying several forms of DDoS attacks. In particular, these works leverage different state-of-the-art network datasets to design robust detection models that can be adapted to constrained network devices, e.g., smartNICs [11] or programmable switches [29], [47].

**Fitting ML-based Models in the Network:** Since programmable data-planes are constrained by nature, multiple attempts have been made to fit learning models in programmable network devices. Here, attempts [5], [6], [11], [17], [47], [48] either choose learning algorithms that can be mapped to programmable data-plane semantics, e.g., match-action tables or transform the trained learning models to suitable representations, e.g., Binary Neural Network (BNN) and split NN. For the former, well-known ML algorithms, i.e., DT and RF, can be reduced and mapped into a set of tables in the programmable devices [5], [17]. For the latter, the majority of the attempts leverage BNN [11], [17], [47] due to their computation simplicity and reduced model size. Further in this regard, another attempt concerns the split of the NN structure, i.e., the neurons, in different devices in the network [12], [48].

**False alarm reduction and collaborative network defense:** False alarms in network security applications are a persistent issue [21], [23]. From the interviews that Vermeer et al. [21] and Alahmadi et al. [22] conducted to network security experts and practitioners, it emerges that handling false alarms in real networking environments is indeed challenging. Several research [23], [24] propose post-hoc methods, i.e., that operate after the alarms are collected, for handling the large rate of false alarms, especially in the context of intrusion detection. Another perspective for dealing with large rates of false alarms is proposing a collaborative mechanism that operates in a coordinated fashion among different devices [25], [26].

# 3. Distributed In-Network Detection

In this section, we describe O'MINE, our in-network DDoS attack detection mechanism. Initially, in Section 3.1, we describe the considered threat model. Then, in Section 3.2, we overview the requirements for designing O'MINE as a novel collaborative DDoS detection mechanism. Then, in Section 3.3, we detail each of its components.

## 3.1. Threat Model

In this paper, we define a DDoS as an attack that comprises multiple coordinated and globally distributed hosts that send high-frequency service requests to the victim. The aim of sending a high number of requests might be: *(i)* saturating the victim's resources, or *(ii)* saturating the victim's network bandwidth with high traffic volume. In both cases, the considered network infrastructure in the victim's network experiences bottlenecks, since the malicious traffic originating from different sources is aggregated. Hence, the detection mechanisms that are deployed in such infrastructures must be flexible and operate fast as the aggregation of the malicious traffic quickly saturates both the local and intermediate path resources. Considering the above, the proposed detection mechanism is designed to be deployed in the victim's network in a collaborative fashion. Such a detection can be used by the network operator to protect the server's resources from malicious traffic originating from other domains or even its own domain. Lastly, the considered threat model does not require the attacking hosts to be coordinated or share particular properties, except for the attack start time. Such a model fits real-world scenarios, where multiple devices, e.g., bots, once activated, simply send frequent requests without further coordination.

## 3.2. Requirements

To design a robust DDoS detection mechanism suitable for deployment in programmable data-plane devices, we stipulate the following requirements for O'MINE:

- R1 *Accurate detection.* Measuring only the accuracy of a ML-based DDoS detection does not necessarily represent its correctness. An estimation of False Positive Rate (FPR) is needed, as it represents the rate of wrongly categorized events. Additionally, relying on a single decision for each network packet restricts the detection precision to the performance of the single model. Instead, a collaborative decision on a per-packet basis helps achieve a more accurate detection.
- R2 *Line rate inference.* Offloading the DDoS attack detection to the data-plane allows running inference on a per-packet basis. However, programmable data-plane devices are designed to process and forward tons of traffic at line rate. Therefore, for our detection mechanism, we require fast inference times that do not interfere with the line rate operation of the network devices to where the detection mechanism is offloaded.
- R3 *Feature engineering feasibility.* As an important step for training accurate ML models, feature engineering should also take into account real-world deployment capabilities. For our detection mechanism, we require multiple-level feature engineering that foresees whether a set of features can be extracted at line rate or require extra computation power that can be offered only with some latency addition.

## 3.3. Design

In this section, we describe the design of O'MINE, our novel collaborative DDoS detection mechanism for programmable data-planes. O'MINE proposes a collaborative decision process on a per-packet basis among several network switches that brings a maximisation of accuracy while keeping a low FPR, fulfilling R1. Additionally, designing O'MINE as a joint decision mechanism with the inference modules running at a data-plane device or an additional network accelerator in the data-plane allows for line rate operation, fulfilling R2. Lastly, O'MINE's design comes with particular attention to the selected training features, not only for the sake of ML model's accuracy but also for the deployment feasibility in real-world scenarios. Figure 1 depicts an overview of the proposed mechanism, encompassing the three phases that we describe subsequently.

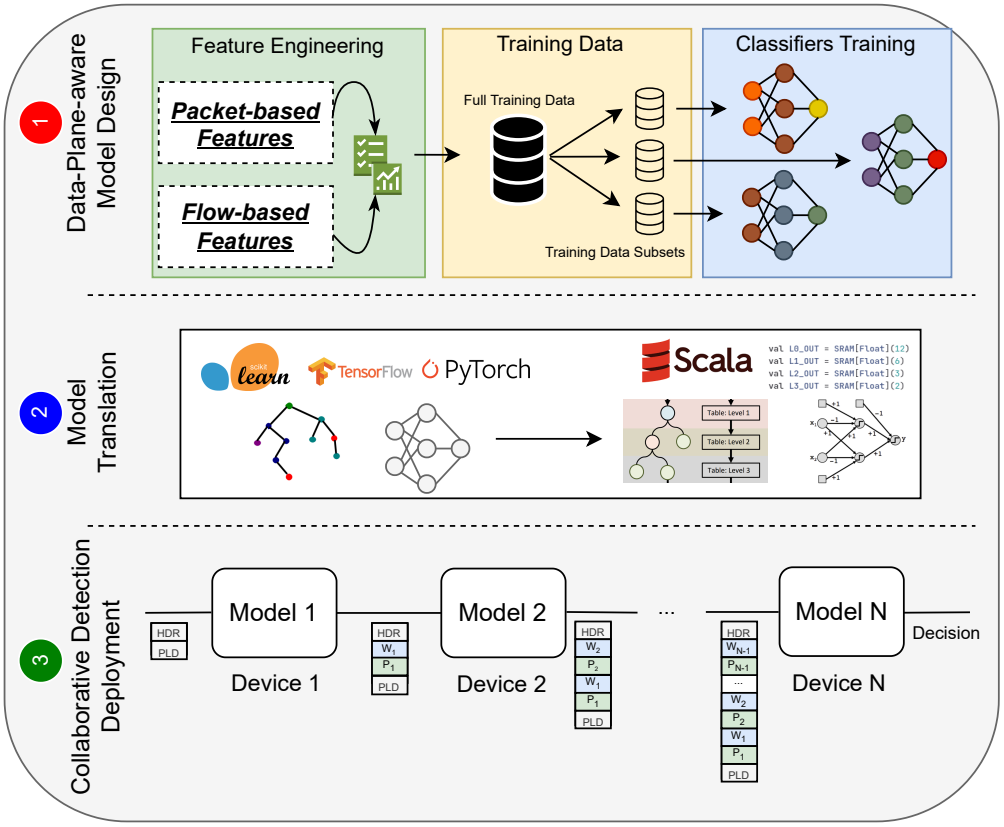


Figure 1: An overview of O'MINE, built over three phases: data-plane-aware model design, model translation and collaborative detection deployment. *Legend:* HDR - Packet Header; PLD - Packet Payload; $W_i$ - Correctness score of switch $i$; and, $P_i$ - Prediction of switch $i$.

1 **Data-plane-aware model design.** In the first step of this phase, O'MINE can extract the data following two feature types, i.e., packet-based and flow-based features, depending on their complexity (as discussed in Section 4.3). The former category includes features that can be obtained from single packets, mainly from the packet header fields. The latter encompasses features combined from multiple packets in the same flow. Secondly, the training data obtained from the previous step is split equally into the number of classifiers that need to be trained for the data-plane devices. Then, a binary classifier is trained for each switch by using different training samples for each of them and the best hyperparameter settings (as detailed in section 4.2. Although in this paper we evaluate O'MINE only on NN models, our mechanism is agnostic to the type of the selected algorithm, e.g., Support Vector Machine (SVM), DT, RF, or NN.

2 **Model translation.** After the design and training of the classifiers, the translation phase is in charge of translating the obtained models to data-plane-aware semantics following predefined types of deployment, i.e., entirely or near data-plane, that the user can choose. In particular, with the entirely data-plane type, we consider the deployment of ML models that can run in the constrained data-plane switches, e.g., DT and RF, as previous works demonstrated [5], [49]. Instead, the near data-plane type refers to the deployment of more complex ML models, e.g., NN, that can not directly be run in programmable data-plane switches due to their computation and resource constraints. Fortunately, state-of-the-art [11], [13] solutions exist that overcome these issues by translating NN to models suitable for programmable network devices, e.g., Tofino switches or FPGAs. Finally, such semantics are deployed into the data-plane devices (as elaborated in section 4.5).

3 **Collaborative detection deployment.** O'MINE's novelty resides in the distribution of the detection mechanism among multiple switches for improved accuracy and lower FPR. To this end, O'MINE relies on two pillars: *(i)* weighted majority voting algorithm and *(ii)* customized switch programs. The former allows for distributed detection where each switch runs a specific model and keeps a weight which represents how accurate the model is during inference at runtime. The intuition of the algorithm is to "reward" the decision of the well-performing classifiers, and "punish" the less accurate classifiers. Deploying such an algorithm requires modification of the switch programs, which O'MINE addresses by providing three types of switch programs.

To better describe the weighted majority voting algo-

rithm, let us denote the classifiers in the underlying network topology as $C_i$ with $i = 1, 2, \ldots, N$ where $N$ denotes the number of switches in the topology. Then, the decision of the $i$-th switch classifier $C_i$ for the packet at time $t$, say $pkt_t$, is denoted by $d_{i,c_j} \in \{0, 1\}$, where $c_j = 1, 2, \ldots, k$ represents the number of maximum $k$ traffic classes. Although we evaluate O'MINE on binary classification task (see section 4.4), its design is agnostic to the number of considered traffic classes. For the binary classification, the decision is $d_{i,c_j} = 1$ if $C_i$ classifies for class $c_j$, otherwise the decision is $d_{i,c_j} = 0$. Additionally, each of the switches computes and updates a correctness score $w_i$ with $i = 1, 2, 3, \ldots, N$. Generally, such scores are initialised with a value of 1 and are updated following Equation (1):

$$w_{i,t} = \begin{cases} w_{i,t-1} + \alpha_i & \text{if } C_i \text{ makes the correct prediction} \\ w_{i,t-1} - \alpha_i & \text{otherwise} \end{cases} \tag{1}$$

where,

$$\alpha_i = \text{\# incorrect predictions for } pkt_t.$$

Here, $\alpha$ is calculated at the last switch and propagated backwards along the path of the prediction. We use the number of incorrect predictions for $pkt_t$ to enable the "punish"-"reward" logic while also bypassing the floating point operation constraints on most of the programmable hardware devices. As an alternative, $\alpha$ can be calculated as a fraction, e.g., Equation (2), by enabling floating point operations [50].

$$\alpha_i = \frac{\text{\# incorrect predictions for } pkt_t}{N} \tag{2}$$

Unless the switch is in the last hop, upon reception of the packet, the switch model classifies the packet and attaches the prediction on the packet header with the respective switch scores. When the packet reaches the last hop switch, its model classifies the packet and thereafter calculates the final decision on the packet by summing the weighted decisions for each traffic class, following Equation (3):

$$\max_{C_i=\{0,1\}} \{\sum_{i=1}^{N} w_i \times \mathbb{1}(d_i = C_i)\} \tag{3}$$

which essentially compares the sum of the weights of switches that predict the benign class with the sum of the weights that predict the attack class. The greatest weighted prediction dominates the decision of the final switch. O'MINE employs a weight limit as a design feature on each switch for the majority voting mechanism, and limiting it to a specific number is an implementation feature that we found experimentally. There are two benefits that the weight limits bring. First, applying a weight limit prevents weight overflow since the mechanism rewards the switches with high accuracy. Second, limiting the weight enhances the accuracy O'MINE by ensuring a more balanced contribution from each switch.

The deployment of the above-described collaborative data-plane detection requires three different functionalities to be executed depending on the switches' location in the prediction sequence. The head switch initialises the packet, the middle switch(es) count(s) and maintain(s) the weights, and the last switch evaluates the majority voting mechanism. Figure 2 depicts the evolution of forward prediction and backward propagation packets over each switch along the path they follow. On the forwarding route, each switch injects the slot for its weight and prediction right after all the headers. The back propagation packets are used to update the weights for the majority voting algorithm. They contain the majority voting prediction label, the number of incorrect predictions, and the prediction of each switch along the original path. Upon receiving such packets, each switch extracts and compares its prediction with the majority voting prediction label. If the prediction matches the majority voting result, the weight stored on each switch is incremented by the number of incorrect predictions made for this packet. If mismatched, the weight is decremented according to Equation (1).

**Head Switch:** it is the first switch in the prediction sequence which performs actions to initialize feature data, such as shifting the ML headers and injecting the header space for the switch count, the weight, and prediction. Then, the head switch forwards the packets to be processed by the ML module. Upon receiving the packets with the prediction information from the ML module, the head switch reads the information stored inside its dedicated register and then writes this weight value to the allocated header space. Lastly, the head switch performs the weight update when receiving a final decision packet from the last switch, as shown in Equation (1), and drops the packet.

**Middle Switch(es):** can consist of one or multiple switches that perform similar operations to the head switch for each received packet. But in contrast to the head switch, the middle switch(es) do(es) not need to shift ML headers but increment(s) the count in the packet so that the packet can reach the next middle switch. Going backwards, each middle switch parses the final decision and the number of correct predictions to increment and update the saved weight accordingly, following Equation (1). Then, the switch erases the header that includes the extracted information from the outgoing header so that the next switch can parse the correct bytes.

**Tail Switch:** is the last and most crucial in the prediction sequence, responsible for two tasks: *(i)* making the final decision and, *(ii)* generating and preparing the back propagation packet for the switches along the paths to update their weights. The number of supported switches on the path needs to be predefined by the user. This is due to the limitation of the programmable network devices that do not naturally support recursion or loops. To calculate the final decision, the tail switch uses Equation (3). Finally, the tail switch emits the original packet to the final destination and the back propagation packet to perform the weight updates using Equation (1). To further reduce network bandwidth usage, we do

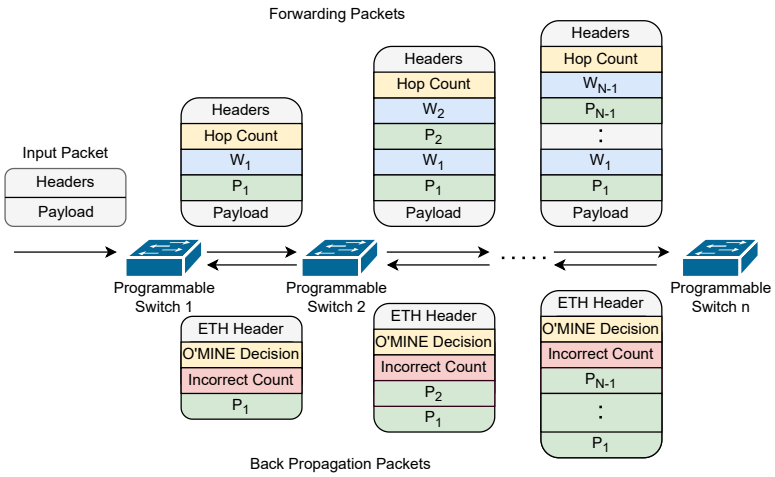


Figure 2: Packet formats evolution after being processed by each switch.

not generate the backward packet if all the switches along the path come to the same decision. This is valid because if all weights are updated with the same value, the majority voting result would not be impacted. In addition, we utilize the `truncate` functionality of the programmable data-plane devices to remove the original payload and only emit the ethernet header with the back propagation header fields to ensure minimal usage bandwidth of our mechanism.

## 4. Evaluation

In this section, we initially describe the experimental set-up in Section 4.1. Then, we present the results from the classifiers engineering in Section 4.2 and feature engineering in Section 4.3, representing the core of the first phase for O'MINE. Lastly, we detail the results from the two-fold evaluation, i.e., software implementation of O'MINE in Section 4.4 and hardware testbed testing in Section 4.5.

### 4.1. Experimental Set-up

In this section, we provide detailed insights regarding the experimental set-up for O'MINE implementation. In particular, the key components of implementation here include the offline classifiers engineering, the customized switch programs, and the implementation of the weighted algorithm. We have built a prototype based on a Tofino switch [51], [52], P4 (data-plane programming language) [40], and bmv2[4] (the software switch) for the evaluation of the switch programs and the weighted majority voting algorithm that is implemented using the *clone* operation provided by the P4 language. Instead, for the classifiers and feature engineering, we use Python3 and the PyTorch[5]. For the O'MINE software implementation, we use a machine with Ubuntu 20.04.5 LTS, with an NVIDIA GeForce RTX 3050 Ti GPU incorporated.

[4]bmv2: https://github.com/p4lang/behavioral-model
[5]PyTorch Library: https://github.com/pytorch/pytorch

### 4.2. Classifiers Engineering

For evaluation purposes, we consider a topology of five switches. Subsequently, we detail the different state-of-the-art network security datasets that we use for training the switches' classifiers. Moreover, to determine the best hyperparameter set-up for the considered model structure, we provide an extensive ablation study.

**Datasets:** Designing O'MINE, we leverage three open source state-of-the-art network security datasets, i.e., NSL-KDD [53], UNSW-NB15 [54], and CICIDS-2017 [55]. The NSL-KDD dataset comprises 41 features collected over benign and malicious traffic related to four attack classes. The UNSW-NB15 dataset includes 49 features gathered from benign samples and traffic related to nine attack classes. Lastly, the CICIDS-2017 dataset encompasses 79 features collected over benign traffic generated in a large-scale campus as well as traffic related to 15 classes of attacks, among which multiple versions of DDoS attacks. From all the datasets, we select only the benign, i.e., Normal, and DDoS traffic samples. Table 1 shows the number of instances before and after this selection. As the table shows, pruning non-DDoS attacks leads to an imbalanced UNSW-NB15 dataset. We balance its classes using upsampling techniques during the training phase in order to ensure the correct training behaviour of the models. Moreover, the CICIDS-2017 contains multiple null and infinite values, which we dropped during the dataset preprocessing.

**Ablation study:** Since O'MINE targets constrained data-plane devices, we consider lightweight fully connected NN that are used for inference in each of the switches. Thus, we design the models to be lightweight by maintaining a shallow and compressed structure, similar to previous works [11], [13] that use three to five layers fully-connected NN. In particular, we consider a five-layer NN and provide an ablation study on its parameters, comprising the batch size and the number of training epochs used. Figure 3 shows the results for the three datasets, namely UNSW-NB15, NSL-KDD, and CICIDS-2017, from left to right, for two hyperparameters, i.e., learning rate and

TABLE 1: Distribution of the class samples for NSL-KDD, UNSW-NB15 and CICIDS-2017 datasets.

| Dataset | All Train Samples | All Test Samples | Selected Train Samples | Selected Test Samples |
|---|---|---|---|---|
| NSL-KDD | Normal: 67.342<br>DDoS: 58.630 | Normal: 9.711<br>DDoS: 12.832 | Normal: 67.342<br>DDoS: 45.927 | Normal: 9.711<br>DDoS: 7.457 |
| UNSW-NB15 | Normal: 37.000<br>DDoS: 45.332 | Normal: 56.000<br>DDoS: 119.341 | Normal: 37.000<br>DDoS: 4.089 | Normal: 56.000<br>DDoS: 12.264 |
| CICIDS-2017 | Normal: 68.386<br>DDoS: 89.636 | Normal: 29.332<br>DDoS: 38.391 | Normal: 68.367<br>DDoS: 89.634 | Normal: 29.319<br>DDoS: 38.391 |

training epochs, while considering batch_size=32. As the results show, the best hyperparameters for the UNSW-NB15 and NSL-KDD datasets are learning_rate=0.001 and epochs=30. Instead, for the CICIDS-2017 dataset, the best hyperparameters are learning_rate=0.001 and epochs=10. Generally, the NSL-KDD dataset led to stable behaviour in all the hyperparameters coordination since it comprises a significant number of training samples. Conversely, the UNSW-NB15 dataset has fewer training samples, therefore leading to slightly decreased accuracy results. In the reminder of the paper, we will use the best combinations of hyperparameters for each dataset to train the NN models.

### 4.3. Feature Engineering

The common features used in network security tasks can often be obtained from a single packet, i.e., packet-based features, or from a flow of packets, i.e., flow-based features. However, the availability of these features in real-world deployment at line rate is often underestimated. In constrained data-plane devices, it is impractical to design models that require computationally expensive input features during inference. For example, one might think of the computation burden for a constrained network device that is required to extract and process an encrypted packet payload. Similarly, collecting statistics from flows of packets at line rate could be an expensive and time-consuming task, disturbing the real-time communication speed. To study how these packet-and-flow-based features influence the classifiers' training outcome, we further define three types of features: *simple*, *flow-mid*, and *flow-complex* features. The *simple* class encompasses features that can be extracted from the packet header at line rate, e.g., protocol type, source or destination ports, packet sizes, and Time To Live (TTL). Instead, we generally group the features that calculate count values under the *flow-mid* type of features that can still be extracted at line rate via special data structures, e.g., count sketches, as previous work demonstrated [49], [56]. Lastly, features that require a complex computation, e.g., mean, standard deviation, maximum, minimum, that cannot be computed via predetermined update rules, are grouped under *flow-complex*.

To understand the phenomena of feature complexity, we train three NNs over the three datasets using all the features. In the first experiment, we use the SHAP library [6] to plot the 20 features with the highest importance score for legitimate traffic (class 0) and DDoS attack (class 1). Subsequently, in the second experiment, we show how the underlying training features from the three classes affect the model accuracy and the false positives ratio. To this end, we trained four models that use different sets of features, i.e., all, simple, flow-mid, and flow-complex. The results depicted in Figures 4 and 5 show that, for UNSW-NB15 and CICIDS-2017, the most impactful features for the classifiers' output are flow-based features, e.g., min, max, count, mean. The reader can refer to Section A for an extensive description of the features' acronyms represented in these figures. Intuitively, being among the most important training features for the classifier outcome, using these features during training would lead to better accuracy compared to simple features. Conversely, for the NSL-KDD dataset, multiple *simple* features already lead to good accuracy results, similar to the *flow-mid* and *flow-complex* classes. Although the selection of features depends on the developer, with this preliminary study we emphasise the importance of not only considering accuracy gain when selecting the features but also examining the feature extraction and processing feasibility in the data-plane context.

### 4.4. O'MINE Detection Performance

For the evaluation of O'MINE, we created a software implementation that comprises the Taurus switch software [13] [7], which enables MapReduce operations for programmable networks. Additionally, we leverage Mininet[8] for the topology specification. Further, for the representation of the trained models, we adopt Spatial [57], and we limit the switches' weight scores to 200. For the traffic generation, we leverage the Python Scapy library[9], and two datasets, i.e., UNSW-NB15 and CICIDS-2017. For the training process, we split the dataset samples into train (75%) and test data (25%). For each dataset, we use the best hyperparameter setting described in Section 4.2 to train the models. All the models are trained on the seven most important features from the SHAP analysis described in Section 4.3. In particular, the features that we use for evaluation for UNSW-NB15 are: $dttl$, $swin$, $dload$, $ct\_dst\_sport\_ltm$, $ct\_dst\_src\_ltm$, $ct\_srv\_dst$, $sttl$. Instead, the used features from CICIDS-2017 are: *destination port*, *bwd packet length min*, *packet length variance*, *URG flag count*, *fwd packet length max*, *avg fwd segment size*, *fwd packet length mean*. The reader can refer to Section A for a detailed elaboration of these features' acronyms. O'MINE utilizes features that can be extracted

[6]SHAP Library: https://shap.readthedocs.io/

[7]Taurus Platform Behavioral Model: https://gitlab.com/dataplane-ai/taurus/platform-bm

[8]Mininet Simulator: https://mininet.org/

[9]Scapy Library: https://scapy.net/

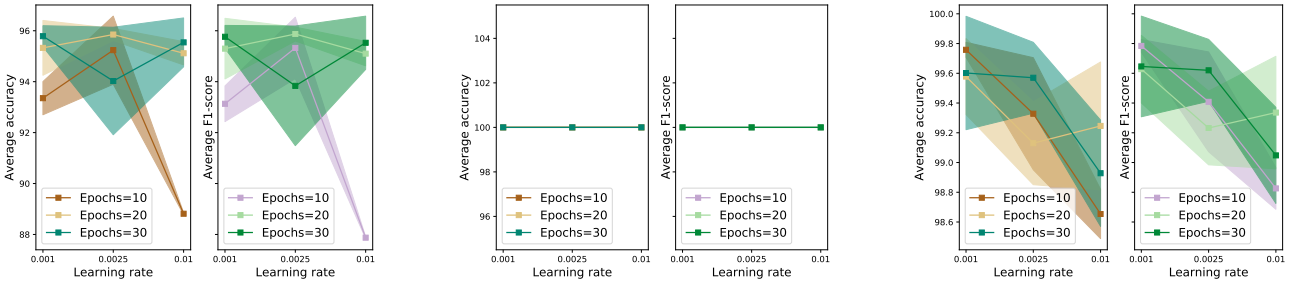


Figure 3: Ablation study for the NN with [8x4x4x4x2] structure trained in 80% of dataset samples and full set of features. The results are computed on the UNSW-NB15, NSL-KDD, and CICIDS2017 datasets, from left to right.

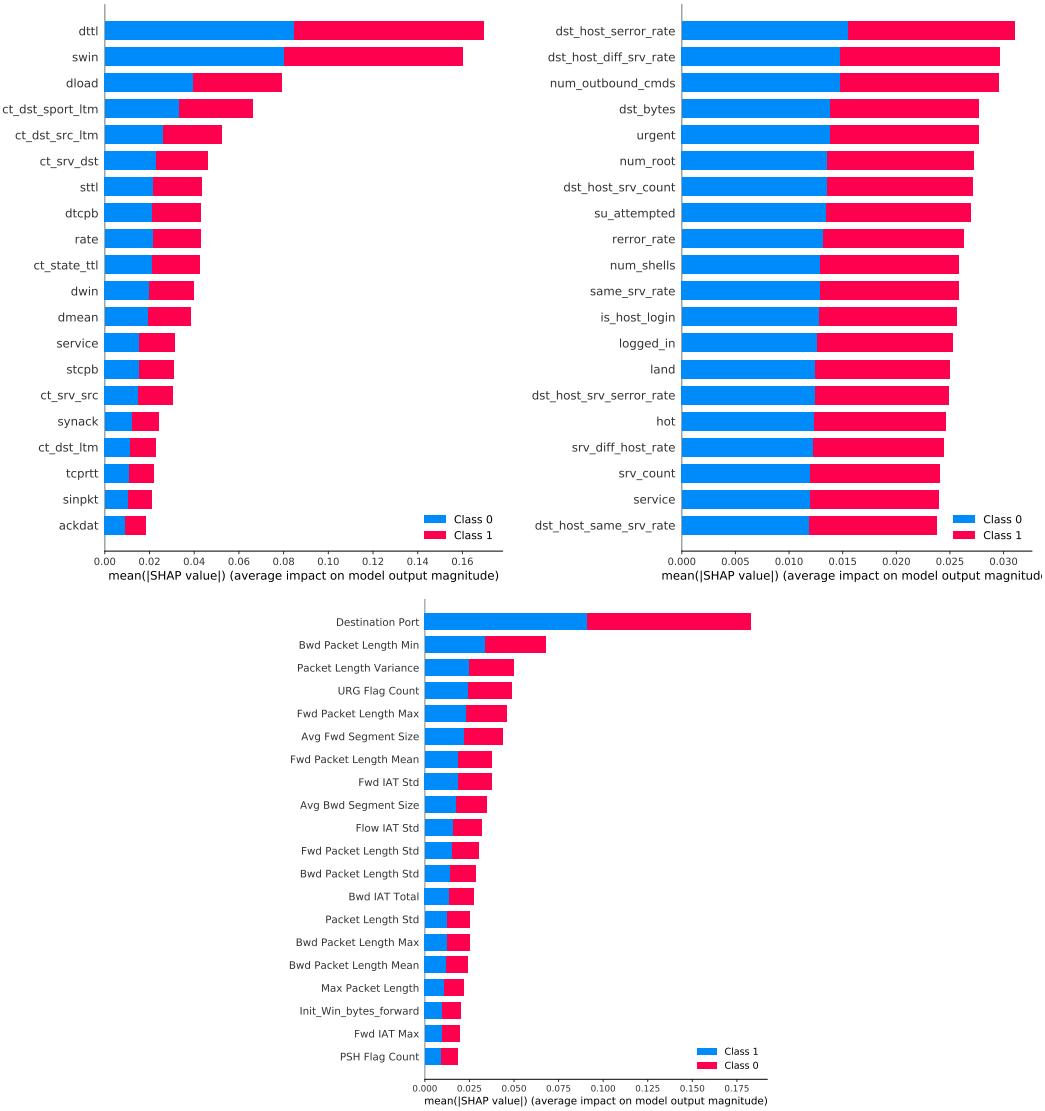


Figure 4: Most important features extracted with SHAP for the [8x4x4x4x2] NN trained over UNSW-NB15, NSL-KDD, and CICIDS-2017, from left to right, respectively.

from the packet header (*sttl*, *dttl*, *swin*), counted via registers (*dload*), or stored in data structures such as sketches (*ct_dst_sport_ltm*, *ct_dst_src_ltm*, *ct_srv_dst*). Subsequently, we evaluate O'MINE in an *(i)* optimal scenario (Section 4.4.1), i.e., the models are trained in full training set; *(ii)* real-world scenario (Section 4.4.2), i.e., the models are trained with scarce data; and, we compare O'MINE with state-of-the-art solutions (Section 4.4.3).

**4.4.1. Optimal Scenario Performance.** In this section, we report the O'MINE performance (accuracy and FPR) for detecting DDoS attacks when the models are trained on the full train set, i.e., 169286 samples for CICIDS and 139501 samples for UNSW-NB15. We consider this an optimal case, as realistically there is not always data avail-

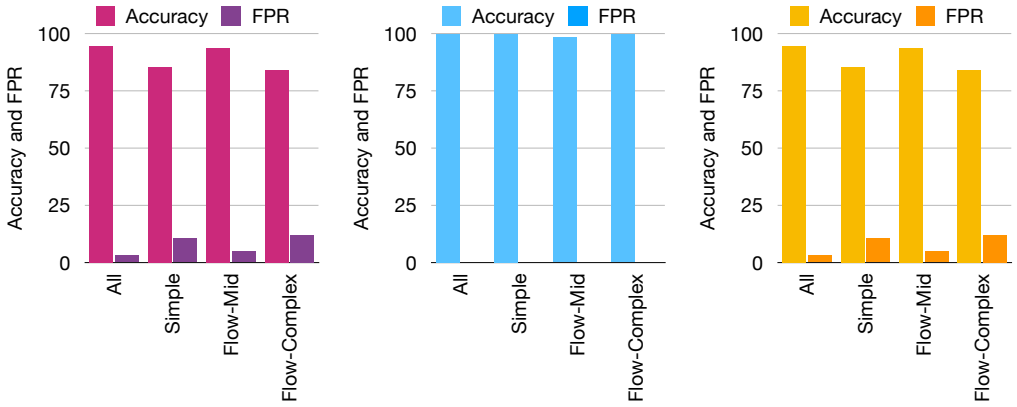


Figure 5: Accuracy and FPR considering four feature sets for UNSW-NB15, NSL-KDD, and CICIDS-2017 datasets, from left to right, respectively.

ability for model training. For testing in Taurus software, we generate traffic from both datasets, i.e., UNSW-NB15 and CICIDS-2017, that will traverse all the five switches considered for the linear topology, using 5000 packets that are randomly sampled from the testing set. However, we also provide O'MINE performance on the full testing set extracted via customised Python scripts. Table 2 shows the accuracy on top and the FPR at the bottom for the models that are trained on the full train set. As expected, O'MINE outperforms the baseline models, i.e., the single model version, over the UNSW-NB15 dataset, showing the benefit of penalizing the worse models that significantly decrease the FPR. Instead, for the CICIDS-2017 datasets, O'MINE achieves the same accuracy as the single model version. These results are aligned with the nature of each dataset. Here, the CICIDS-2017 dataset is simple as the DDoS attack and benign traffic classes are easily separable. Instead, the UNSW-NB15 dataset is more complex, especially due to its imbalanced nature. Therefore, O'MINE in the latter case brings significant improvements.

**4.4.2. Real-world Scenario Performance.** A more realistic scenario involves scarcity in the training data. To check O'MINE's performance in this scenario, we further sample the train data for both datasets to create more realistic training for switches models. Table 3 shows how the accuracy and FPR changes for different sampling sizes. It can be noticed that the CICIDS-2017 dataset achieves very good performance even with a small number of training samples, demonstrating that it is a well-balanced dataset that decently separates the DDoS class. On the other hand, the performance for the UNSW-NB15 dataset drops when a small number of samples is used during training. For our experiments, we sample only 0.5% of CICIDS-2017 training data, corresponding to 846 samples and 1% of UNSW-NB15 training data, corresponding to 1395 samples. Table 4 depicts the results for O'MINE when models are trained on scarse CICIDS-2017 data. O'MINE is able to outperform the single switches. However, this drastic sampling of training data actually results in performance degradation. Instead, for the UNSW-NB15 dataset, we not only trained with scarce data, but we also intentionally introduced a faulty model, i.e., a model that is performing very poorly, less than random guessing. The depicted results in Table 5 show that O'MINE is able to outperform all the baselines, achieving only a 0.70% FPR when one of the switches actually has 100% of FPR.

**4.4.3. Comparison with SOTA.** Table 6 represents details on prior DDoS detection mechanisms that use programmable devices' resources and/or network accelerators [11], [12], [29], [47], and general CPUs [58]–[60]. In particular, the table depicts the considered algorithms, datasets, whether the source code is available, and deployment resources. For fair analysis, we compare O'MINE with N3IC [11] since it deals with the challenge of deploying NN in today's network accelerators, similarly to what O'MINE considers for deployment. We additionally extend the comparison to state-of-the-art ML-based DDoS detection mechanisms that use general CPUs, i.e., Zhou et al. [58], Stiawan et al. [59] and Gu et al. [60].

**N3IC VS O'MINE:** N3IC leverages BNNs in various configurations, i.e., [32x16x2], [64x32x2] and [128x64x2], for attack detection on the USNW-NB15 dataset. Their detection accuracy ranges from 85% (for the smallest and medium BNN) to 91% (for the largest BNN), which is lower than O'MINE's accuracy (95.40%). Additionally, all their BNN models reach around 15.9% FPR. Such a high FPR compared to the 1.19% for O'MINE confirms the superiority of a coordinated detection. Additionally, O'MINE considers a slightly deeper NN, i.e., 4 hidden layers instead of 3, but with fewer neurons per layer. Yet, O'MINE is able to reach better performance, i.e., accuracy and FPR, while using smaller models. For a second equal comparison, we leveraged state-of-the-art techniques that enable the NN quantization and binarization to transform our trained NN, similar to N3IC. Quantization simplifies the NN model by reducing the number of bits for the representation of the activations and parameters without sacrificing much of the model accuracy [61]. Instead, binarization is an extreme form of quantization that enables additional model size reduction by using single-bit input and weights, which drastically reduces memory consumption while also enabling data-plane-friendly operations. We adopt the Xilinx Brevitas[10]

[10]Brevitas Library: https://doi.org/10.5281/zenodo.3333552

TABLE 2: **Full training dataset:** O'MINE detection accuracy (top) and FPR (bottom) for five switches topology trained over the full UNSW-NB15 and CICIDS-2017 datasets.

| Test Dataset | Test Size | Switch 1 | Switch 2 | Switch 3 | Switch 4 | Switch 5 | **O'MINE** |
|---|---|---|---|---|---|---|---|
| UNSW-NB15 | 14038 samples | 85.70%<br>11.39% | 88.60%<br>8.35% | 88.80%<br>8.85% | 73.40%<br>24.15% | 88.30%<br>8.89% | **95.4%**<br>**1.19%** |
| | 5000 samples | 66.64%<br>28.48% | 83.62%<br>7.82% | 84.28%<br>9.13% | 75.88%<br>17.43% | 83.54%<br>8.59% | **90.50%**<br>**0.22%** |
| CICIDS-2017 | 56424 samples | 98.28%<br>3.91% | 98.28%<br>3.91% | 98.28%<br>3.91% | 98.28%<br>3.91% | 98.28%<br>3.91% | **98.28%**<br>**3.91%** |
| | 5000 samples | 97.82%<br>4.32% | 97.82%<br>4.32% | 97.82%<br>4.32% | 97.82%<br>4.32% | 97.82%<br>4.32% | **97.82%**<br>**4.32%** |

TABLE 3: Accuracy (top) and FPR (bottom) of our model over UNSW-NB15 and CICIDS-2017 datasets with different percentages of train sampling.

| Dataset | 100% | 60% | 20% | 5% | 1% |
|---|---|---|---|---|---|
| UNSW-NB15 | 96.19%<br>4.33% | 94.48%<br>3.47% | 88.83%<br>2.68% | 93.31%<br>4.40% | 86.97%<br>6.90% |
| CICIDS-2017 | 98.93%<br>0.14% | 99.80%<br>0.17% | 99.64%<br>0.15% | 99.13%<br>0.20% | 99.01%<br>2.05% |

library to evaluate the performance of the O'MINE models in a quantized and binarized fashion. For the quantization procedure, we make use of the Quantization Aware Training (QAT) from Brevitas to quantize all weights, biases, and activations during both the forward and backward passes of training. That is, the NN is transformed from Float32 values to Int8 values. Similarly, the binarization procedure leads to a BNN that ensures the single-bit quantization for weights, biases and activations. In Figure 6, we plot the accuracy and memory consumption for the oracle NN, its quantized version, and its BNN for the simple, flow-mid, and flow-complex features, that are represented as floats or binaries, respectively. For the float feature representation, both the quantized and binarized O'MINE models achieve up to 85% of accuracy (equal to the smallest N3IC model), with a size around 150 Bytes. Instead, for the binary feature representation, our models achieve 75% to 90% of accuracy for binarized and quantized models, respectively.

**ML-based mechanisms VS O'MINE:** Additionally, we compare O'MINE to state-of-the-art ML-based network intrusion detection systems [58]–[60]. For a fair comparison with O'MINE, we use the set of seven most important features (as described in Section 4.3) and the full training set for each dataset. Zhou et al. [58] reaches an accuracy of 97.02% and lower FPR 6.89%, quite double O'MINE FPR. Instead, Stiawan et al. [59] obtains a maximum accuracy of 95.83% and the lower FPR of 9.64% with RF model. However, O'MINE outperforms this baseline, especially since it achieves 6% less FPR. Also, with Random Tree (RT) model, the performance degrades to 75.11% of accuracy and 57.57% of FPR. Lastly, Gu et al. [60] reaches a 96.85% and % of accuracy, and 7.21% and 73.06% of 27.55FPR, for the CICIDS-2017 and UNSW-NB15, respectively. O'MINE outperforms the baseline in both datasets, especially for UNSW-NB15, confirming the superiority of our collaborative mechanism.

**Non-ML-based mechanisms VS O'MINE:** Lastly, we briefly compare O'MINE with non-ML DDoS detection mechanisms, i.e., mod_evasive, fail2ban and SYN cookies. To start with, mod_evasive is a module that works by creating and maintaining lists of IP addresses that should be banned. However, scaling such a solution to large networks introduces management complexities (e.g., proper configuration for large networks to avoid false positives) and resource consumption (i.e., the overhead of maintaining dynamic tables for numerous IP addresses can become rather significant). Similarly, fail2ban is a server-based monitoring mechanism that checks for malicious IPs that have suspicious activities or multiple failed login attempts. As such, it introduces similar adoption challenges to mod_evasive. Lastly, SYN cookies is a server-based mechanism that specifically works for the detection of SYN flooding, a type of DDoS attack. Differently from the mechanisms mentioned earlier, O'MINE is designed to work in a distributed fashion involving multiple networking nodes, and it is able to scale as shown in Section 4.6.

**4.4.4. Timing.** For the timing evaluation, we run experiments to measure the performance of the back-propagation method proposed by O'MINE. To measure the latency of one packet to traverse back from the last switch of the collaborative weighted voting mechanism to the first switch, we collect the `egress_global_timestamp` metadata provided inside the v1model at the last switch and the `ingress_global_timestamp` at the first switch. Then we calculate the difference between the two values to check the time taken from the egress of the last switch and the ingress of the first switch, and the result shows that the back-propagation latency takes around *47 ms* to traverse through the switches and update the weights accordingly. Such a number does not affect the detection accuracy since the switches are able to run inference on incoming traffic independently. Moreover, in the following section, we show that such timing is negligible when measured in a real-world testbed, reaching levels of $\mu$s.

## 4.5. O'MINE Hardware Performance

For the O'MINE end-to-end evaluation in hardware, we leverage a Tofino Wedge100BF-32x switch [51] and a Xilinx Alevo U250 FPGA[11]. The Tofino switch is used to bypass traffic through the FPGA that is used to implement the ML models. While here we adopt an FPGA for testing NNs, O'MINE detection is agnostic to the type of model

[11]Xilinx Alevo U250 Data Center Accelerator Card: https://www.xilinx.com/products/boards-and-kits/alveo/u250.html

TABLE 4: **Scarse training dataset:** O'MINE detection accuracy (top) and FPR (bottom) for five switches topology trained over the CICIDS-2017 dataset with 0.5% of sampling on the training data, i.e., 846 samples.

| Test Dataset | Test Size | Switch 1 | Switch 2 | Switch 3 | Switch 4 | Switch 5 | **O'MINE** |
|---|---|---|---|---|---|---|---|
| CICIDS-2017 | 56424 samples | 75.45%<br>56.77% | 90.37%<br>22.19% | 83.72%<br>37.57% | 91.78%<br>18.93% | 90.72%<br>21.37% | **92.51%**<br>**17.25%** |
| | 5000 samples | 80.76%<br>38.44% | 90.66%<br>18.64% | 73.46%<br>53.04% | 89.70%<br>20.56% | 70.32%<br>59.32% | **91.48%**<br>**17.00%** |

TABLE 5: **Scarse training dataset and faulty models:** O'MINE detection accuracy (top) and FPR (bottom) for five switches topology trained over the UNSW-NB15 datasets with 1% of sampling on the training data, i.e., 1395 samples.

| Test Dataset | Test Size | Switch 1 | Switch 2 | Switch 3 | Switch 4 | Switch 5 | **O'MINE** |
|---|---|---|---|---|---|---|---|
| UNSW-NB15 | 14038 samples | 63.68%<br>34.14% | 81.46%<br>15.74% | 88.58%<br>8.67% | 3.37%<br>100.00% | 88.41%<br>8.84% | **95.99%**<br>**0.70%** |
| | 5000 samples | 83.46%<br>7.97% | 83.86%<br>8.41% | 9.48%<br>100.00% | 93.74%<br>8.55% | 80.56%<br>11.18% | **86.08%**<br>**5.08%** |

TABLE 6: State-of-the-art on DDoS detection in programmable data-plane devices and general CPUs. Legend: Binary Neural Network (BNN); Random Forest (RF); Decision Tree (DT); Naive Bayes (NB); Random Tree (RT); Support Vector Machine (SVM).

| Paper | Algorithms | Datasets | Source Code Available | Deployment |
|---|---|---|---|---|
| [11] | [32x16x2] BNN<br>[64x32x2] BNN<br>[128x64x2] BNN | UNSW-NB15 | Yes | SmartNIC |
| [12] | Split NN | IoTNIDS | No | P4 Switch |
| [29] | XGBoost | Custom | Yes | P4 Switch |
| [47] | BNN in FL fashion | CICIDS-2017<br>ISCX Botnet 2014 | Yes | P4 Switch |
| [58] | RF, DT | CICIDS-2017 | No | General CPU |
| [59] | RF, RT, NB | CICIDS-2017 | No | General CPU |
| [60] | SVM | UNSW-NB15<br>CICIDS-2017 | No | General CPU |

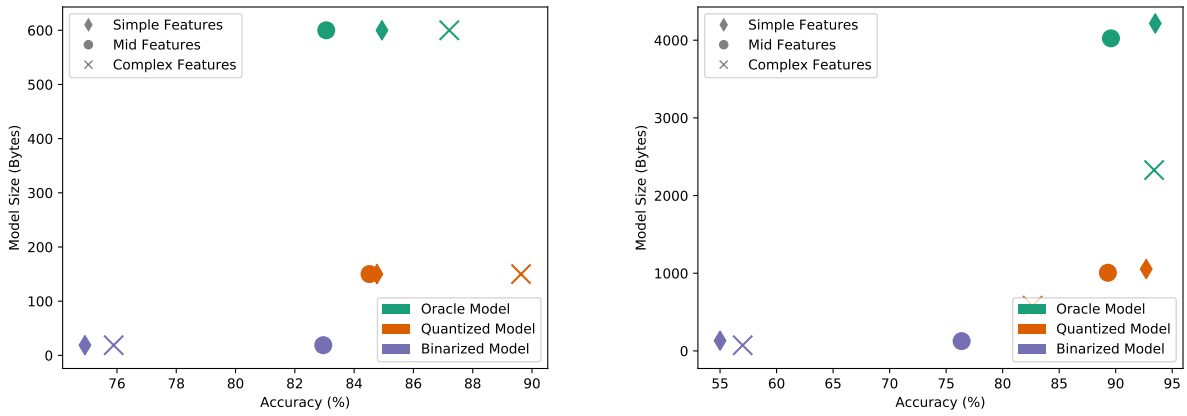


Figure 6: Accuracy vs model size for simple, flow-mid, and flow-complex set of features, which are represented as floats and binary, from left to right, extracted from the UNSW-NB15 dataset.

used, e.g., state-of-the-art ML models that can run in the switch or lightweight representations of NN are equally valid. To test O'MINE in the data-plane, we generate traffic containing the testing set and route each packet through the switch and the FPGA. Initially, the switch preprocesses the features across the flowing packets. Then, the obtained features are transformed into fixed-point numbers that are then forwarded to the FPGA, where the model resides. The underlying model makes a decision on whether the packet is benign or a DDoS. Going back to the switch, the packet traverses a postprocessing MAT, which interprets the decision if it is the last switch. Otherwise, the packet is forwarded to the next switch.

**4.5.1. Performance and Resource Consumption.** To evaluate the detection performance and resource consumption on a per-packet basis, we select one model for each dataset from one of the switches and run it in the FPGA. As shown in Figure 7, the model trained on the UNSW-NB15 dataset maintains its offline accuracy (90.04%) and FPR (0.73%) on the hardware FPGA testing. Similarly, the CICIDS-2017 model keeps its offline accuracy (97.94%) and FPR (4.04%), as depicted in Figure 8. Further, according to the results shown in Figure 9, the models used by O'MINE consume only 6% of the available Look-Up Tables (LUT) of the FPGA. Further, the consumed Flip Flops and BRAM reach 4% and URAM only 1%, showing the efficiency of the trained models. Compared to N3IC [11] that reports 12.6% LUT and 18.8% BRAM consumption for their inference system on an FPGA testbed and simple feature extraction, O'MINE is able to achieve equal or better accuracy results, lower FPR (as showed in Section 4.4.3) while consuming less resources.

## 4.6. Scalability

Evaluating O'MINE scalability, we focus on three main aspects that are relevant as the network topology scales, i.e., the necessary time to back-propagate the final decision, the overhead introduced in the packet header in the forwarding path and the variation of the performance as the packet can traverse different paths—thus, different model combinations. O'MINE evaluations on timing (Section 4.6.1), overhead (Section 4.6.2) performance (Section 4.6.3) are realized on a tree-like topology, as shown in Figure 10.

**4.6.1. Timing.** To further demonstrate the feasibility of our back-propagating algorithm, we implemented it using Intel Tofino hardware [51]. In the hardware implementation, the same amount of parser states, as well as the register access, were used to get as precise latency information as possible. The experiments were conducted across three BF2556X-1T [52] switches with 10Gbps link speed. The measurement packets traverse through the 10Gbps link and the switch data-plane various times to simulate the different number of switches involved in majority voting. The measurements indicate the end-to-end latency from the tail switch to the head switch. In addition, we measure how link saturation affects the transmission latency of the back propagation mechanism. Figure 11 illustrates the results of the average back propagation time (in the range of microseconds) and percentage of link saturation. The results indicate that as the number of switches involved increases and the link saturation intensifies, the back propagation latency also grows. However, despite the expected linear growth with more switches involved, the overall latency of the back propagation mechanism remains within the microseconds range. This demonstrates that O'MINE's back propagation mechanism is still scalable, ensuring that the first switch can receive its update within a reasonable time frame.

**4.6.2. Overhead.** As the network scales, it is of paramount importance to evaluate how much overhead O'MINE introduces. For the forwarding path, since each switch inserts its weight and prediction into the packet header, for the case of an 8-bit weight value, each switch hop introduces an overhead of two extra bytes. With the capability of hardware programmable data-plane devices, such as Tofino switches [51], that support Jumbo Frames up to 9000 bytes, there is sufficient header space for header data to be inserted by theoretically thousands of switches for a network deployed with many programmable data-plane switches.

**4.6.3. Performance.** To evaluate O'MINE's scalability in terms of performance in case the packet can follow multiple paths, we make the case of a packet going from Sender 1 to Server 1 in Figure 10. Following the tree topology representation, the packet can follow a primary path of five hops, i.e., meaning that the packet traverses five switches, or an alternate path of nine hops, i.e., meaning that the packet traverses nine switches. As the packet is initiated at the sender, it follows one of the paths with a specific probability. Such a probability value is not known upfront and depends on the specific scenario. Therefore, we extend our evaluation for O'MINE by studying its behaviour at the variation of such a probability. Therefore, in our experiment, we are interested in showing how O'MINE is able to correctly update the scores on the switches even if the packet traverses different paths, allowing us to keep up with accuracy and FPR. Figure 12 depicts the results of such an experiment. As expected, in the two extremities, i.e., path probability is zero or path probability is one, O'MINE has the exact same accuracy of one path. As the probability varies and reaches 0.5, O'MINE obtains the lower accuracy (higher FPR) as represented by the variable path line in Figure 12. Such an effect is expected since the updates on the switches scores happen only partially, i.e., half of the tested packets follow one path and are assigned a decision with a set of models, while the other half follows the second path. Subsequently, as the probability increases, the accuracy goes up (and FPR down) again. The same trend can be observed for the models trained on the scarce data setting. Concluding, with this experiment, we showed that O'MINE can easily keep up with the expected performance even in multi-path scenarios that are verifiable in real-world scenarios.

# 5. Discussion and Future Directions

In this paper, we have proposed O'MINE, a novel collaborative mechanism that brings notable DDoS detection improvement while also overcoming some shortcomings. In the following, we discuss some relevant aspects for feature development of O'MINE.

## 5.1. Online Models Update

Similarly to state-of-the-art ML-based detection techniques, O'MINE focuses on training static models that are efficiently transferred to programmable network devices. Generally, this is a widely discussed issue in the ML community, especially because of the possible silent degradation of the models once deployed in real-world scenarios. Here, a fundamental cause is the drift in the real data distribution compared to the training data distribution [62], [63]. Ways of mitigating the data drift concept that we might adopt for O'MINE in the future encompass transfer

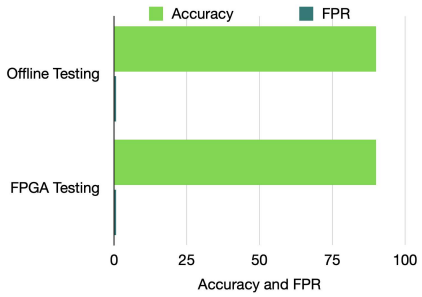


Figure 7: Detection performance for the UNSW-NB15 model.

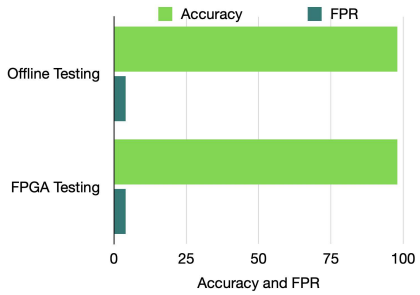


Figure 8: Detection performance for the CICIDS-2017 model.

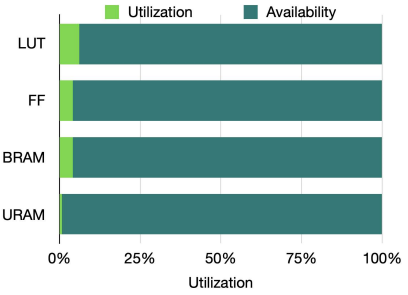


Figure 9: Hardware resource consumption.

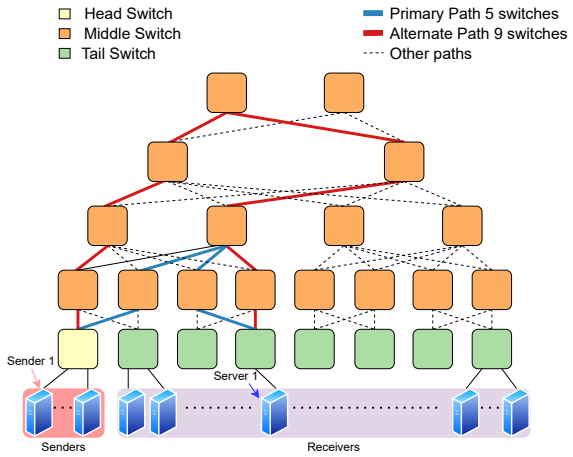


Figure 10: Tree topology that encompasses 24 switches used for scalability evaluation.

learning [64] and online retraining on unsupervised new data [65], [66].

### 5.2. Real-world deployment

DDoS detection is among those network traffic analysis tasks that are bound to real-world deployment. In particular, deploying O'MINE would require the network operator to deploy and eventually update different models residing on the switches over the underlying infrastructure. Itt is of interest for future works to investigate the full potential of O'MINE by taking larger networks into consideration, e.g., the optimisation of the number of models involved, online dynamic machine learning model tuning where all the models reach optimal capabilities on certain detection task, and dynamic adapting of the network for various attack patterns.

### 5.3. Evasion attacks

These are indeed a threat to ML-based security systems and have become a significant area of research due to their potential to undermine the reliability and security of these systems. These attacks involve subtly altering input data to deceive ML models into making incorrect predictions. However, as a collaborative mechanism where multiple models participate in the inference process, O'MINE can mitigate this issue by training models on different sets of features. This ensures that accuracy is not compromised if a specific feature is perturbed. For instance, a vulnerable model relying on the TTL value can be counterbalanced by another model that relies on different features within the weighted collaboration mechanism. Another aspect that makes O'MINE robust against evasion attacks is the insertion of the model prediction and the switches' weights into the packet header. Here, attackers may attempt to manipulate packet size to be exactly 1500 bytes (the standard ethernet MTU) to prevent header insertion. As mentioned earlier, hardware programmable data-plane devices, such as Tofino switches [51], support Jumbo Frames with an increased MTU of up to 9000 bytes. This circumvents such attacks by introducing sufficient space for header insertion inside the network.

## 6. Conclusion

Current state-of-the-art DDoS detection research is mainly driven by the design of customized models that can be embedded in resource-constrained programmable data-plane devices. Although valid, such solutions do not fully cover all crucial elements in DDoS detection, such as accurate detection that considers also minimization of FPR, extraction of traffic features that feed the model with a real-world deployment feasibility by design, and not compromising the line rate operation. This paper proposes O'MINE, a novel collaborative DDoS detection among different switches that each runs inference on various lightweight models. This orchestrated detection mechanism enables collaborative detection, where a single packet is examined by multiple switches before a decision is made, leading to accuracy maximisation and reduction of the FPR. Via software we demonstrate that O'MINE can detect DDoS attacks with high accuracy ($\approx$98% and $\approx$96% with full and scarce training data, respectively) and low FPR ($\approx$0.22% and $\approx$0.72% with full and scarce training data, respectively), outperforming the state-of-the-art. Lastly, our hardware experiments show that O'MINE can operate at line rate with up to $\approx$97% of accuracy and as low as $\approx$0.73% of FPR, showing promising results for new data-plane DDoS detection mechanisms.

## Acknowledgments

We would like to thank our anonymous reviewers for their invaluable feedback, which significantly enhanced

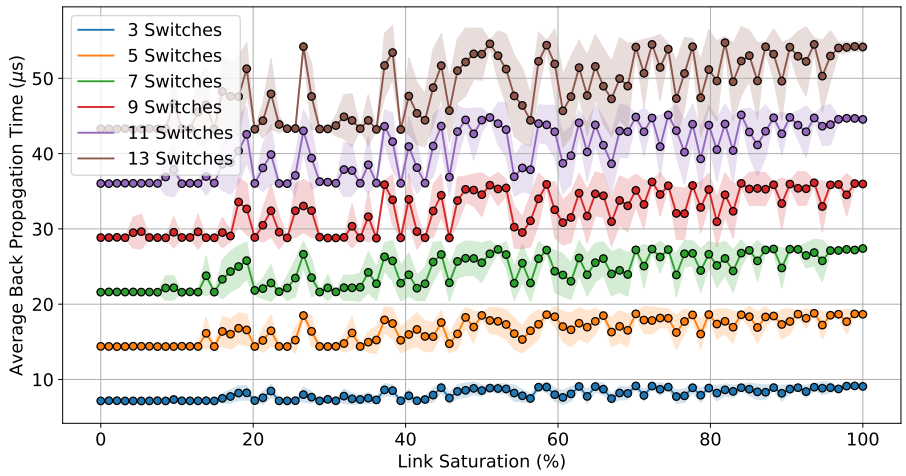


Figure 11: End-to-end weight update latency results for the back propagation mechanism considering different numbers of switches involved and different link saturation. Each data point is computed as an average of the ten iterations, and the shade depicts the standard deviation of each data point.

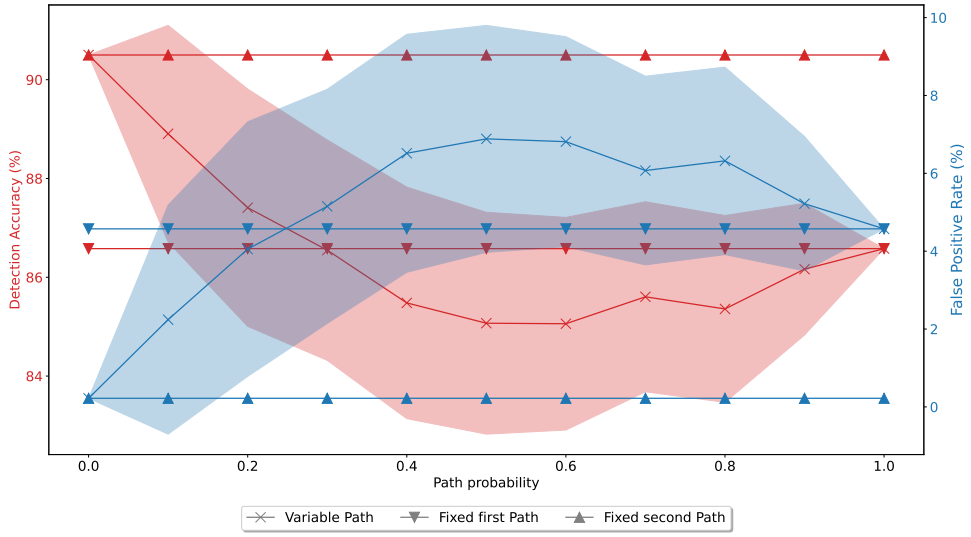


Figure 12: Accuracy and FPR for models trained on full UNSW-NB15 training set in case the packet can follow a variable path between two different paths, i.e., the primary path with 5 switches (fixed second path) or the alternate path with 9 switches (fixed first path).

the quality of this paper. We also thank Dr. Andrea Agiollo, Adrian Zapletal and all the other colleagues across different affiliations that actively engaged in discussions regarding O'MINE's improvement. This research was supported by: "SERICS" (PE00000014) under the NRRP MUR program funded by the EU-NGEU; NSF awards CAREER2338034; and CNS-2211381.

## References

[1] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A survey on software-defined networking," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 1, pp. 27–51, 2015.

[2] N. P. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "HULA: scalable load balancing using programmable data planes," in *Proceedings of the Symposium on SDN Research, SOSR 2016, Santa Clara, CA, USA, March 14 - 15, 2016*. ACM, 2016, p. 10.

[3] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22, 2014*. ACM, 2014, pp. 503–514.

[4] 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 *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[5] Z. Xiong and N. Zilberman, "Do switches dream of machine learning?: Toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks, HotNets 2019, Princeton, NJ, USA, November 13-15, 2019*. ACM, 2019, pp. 25–33.

[6] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, A. Akella, and M. M. Swift, "ATP: in-network aggregation for multi-tenant learning," in *18th USENIX Symposium on Networked Systems Design and*

*Implementation, NSDI 2021, April 12-14, 2021*. USENIX Association, 2021, pp. 741–761.

[7] A. Sapio, M. Canini, C. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtárik, "Scaling distributed machine learning with in-network aggregation," in *18th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2021, April 12-14, 2021*. USENIX Association, 2021, pp. 785–808.

[8] F. Y. Yan, J. Ma, G. D. Hill, D. Raghavan, R. S. Wahby, P. A. Levis, and K. Winstein, "Pantheon: the training ground for internet congestion-control research," in *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*. USENIX Association, 2018, pp. 731–743.

[9] K. Winstein and H. Balakrishnan, "TCP ex machina: computer-generated congestion control," in *ACM SIGCOMM 2013 Conference, SIGCOMM 2013, Hong Kong, August 12-16, 2013*. ACM, 2013, pp. 123–134.

[10] E. Bardhi, M. Conti, and R. Lazzeretti, "Is ai a trick or t (h) reat for securing programmable data planes?" *IEEE Network*, 2024.

[11] G. Siracusano, S. Galea, D. Sanvito, M. Malekzadeh, G. Antichi, P. Costa, H. Haddadi, and R. Bifulco, "Re-architecting traffic analysis with neural network interface cards," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 513–533.

[12] T. Dao and H. Lee, "Jointnids: Efficient joint traffic management for on-device network intrusion detection," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 12, pp. 13 254–13 265, 2022.

[13] T. Swamy, A. Rucker, M. Shahbaz, I. Gaur, and K. Olukotun, "Taurus: a data plane architecture for per-packet ML," in *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*. ACM, 2022, pp. 1099–1114.

[14] A. Agiollo, E. Bardhi, M. Conti, R. Lazzeretti, E. Losiouk, and A. Omicini, "GNN4IFA: Interest flooding attack detection with graph neural networks," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023, pp. 615–630.

[15] E. S. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. M. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, M. Abeydeera, L. Adams, H. Angepat, C. Boehn, D. Chiou, O. Firestein, A. Forin, K. S. Gatlin, M. Ghandi, S. Heil, K. Holohan, A. E. Husseini, T. Juhász, K. Kagi, R. Kovvuri, S. Lanka, F. van Megen, D. Mukhortov, P. Patel, B. Perez, A. Rapsang, S. K. Reinhardt, B. Rouhani, A. Sapek, R. Seera, S. Shekar, B. Sridharan, G. Weisz, L. Woods, P. Y. Xiao, D. Zhang, R. Zhao, and D. Burger, "Serving DNNs in real time at datacenter scale with project brainwave," *IEEE Micro*, vol. 38, no. 2, pp. 8–20, 2018.

[16] Â. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time DDoS attack detection to programmable data planes," in *IFIP/IEEE International Symposium on Integrated Network Management, IM 2019, Washington, DC, USA, April 09-11, 2019*. IFIP, 2019, pp. 19–27.

[17] D. Sanvito, G. Siracusano, and R. Bifulco, "Can the network be the AI accelerator?" in *Proceedings of the 2018 Morning Workshop on In-Network Computing*, 2018, pp. 20–25.

[18] D. C. Li, M. R. Maulana, and L.-D. Chou, "Nnsplit-søren: Supporting the model implementation of large neural networks in a programmable data plane," *Computer Networks*, vol. 222, p. 109537, 2023.

[19] H. Hu, J. Liu, Y. Zhang, Y. Liu, X. Xu, and J. Tan, "Attack scenario reconstruction approach using attack graph and alert data mining," *Journal of Information Security and Applications*, vol. 54, p. 102522, 2020.

[20] A. Presekal, A. Stefanov, V. S. Rajkumar, and P. Palensky, "Attack graph model for cyber-physical power systems using hybrid deep learning," *IEEE Trans. Smart Grid*, vol. 14, no. 5, pp. 4007–4020, 2023.

[21] M. Vermeer, N. Kadenko, M. van Eeten, C. Gañán, and S. Parkin, "Alert alchemy: Soc workflows and decisions in the management of nids rules," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2770–2784.

[22] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: a qualitative study of SOC analysts' perspectives on security alarms," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2783–2800.

[23] C. Fu, Q. Li, K. Xu, and J. Wu, "Point cloud analysis for ml-based malicious traffic detection: Reducing majorities of false positive alarms," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1005–1019.

[24] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *network and distributed systems security symposium*, 2019.

[25] H. Zhou, S. Hong, Y. Liu, X. Luo, W. Li, and G. Gu, "Mew: Enabling large-scale and dynamic link-flooding defenses on programmable switches," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 3178–3192.

[26] D. Wagner, D. Kopp, M. Wichtlhuber, C. Dietzel, O. Hohlfeld, G. Smaragdakis, and A. Feldmann, "United we stand: Collaborative detection and mitigation of amplification DDoS attacks at scale," in *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, 2021, pp. 970–987.

[27] G. Apruzzese, M. Colajanni, L. Ferretti, A. Guido, and M. Marchetti, "On the effectiveness of machine and deep learning for cyber security," in *2018 10th international conference on cyber Conflict (CyCon)*. IEEE, 2018, pp. 371–390.

[28] G. Apruzzese, P. Laskov, and J. Schneider, "Sok: Pragmatic assessment of machine learning for network intrusion detection," in *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2023, pp. 592–614.

[29] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023, pp. 6203–6220.

[30] S. Dong, K. Abbas, and R. Jain, "A survey on distributed denial of service (DDoS) attacks in SDN and cloud computing environments," *IEEE Access*, vol. 7, pp. 80 813–80 828, 2019.

[31] B. de Neira Anderson, B. Kantarci, and M. Nogueira, "Distributed denial of service attack prediction: Challenges, open issues and opportunities," *Comput. Networks*, vol. 222, p. 109553, 2023.

[32] C. Douligeris and A. Mitrokotsa, "DDoS attacks and defense mechanisms: classification and state-of-the-art," *Computer Networks*, vol. 44, no. 5, pp. 643–666, 2004.

[33] J. Xing, W. Wu, and A. Chen, "Architecting programmable data plane defenses into the network with fastflex," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks, HotNets 2019, Princeton, NJ, USA, November 13-15, 2019*. ACM, 2019, pp. 161–169.

[34] E. Haleplidis, K. Pentikousis, S. Denazis, J. H. Salim, D. Meyer, and O. Koufopavlou, "Software-Defined Networking (SDN): Layers and Architecture Terminology," RFC 7426, Jan. 2015.

[35] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. New York, NY, USA: Association for Computing Machinery, 2013, p. 99–110.

[36] C. Ji and F. Kuipers, "State4: State-preserving reconfiguration of P4-programmable switches," in *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, 2023, pp. 134–142.

[37] B. Turkovic, J. Oostenbrink, F. Kuipers, I. Keslassy, and A. Orda, "Sequential zeroing: Online heavy-hitter detection on programmable hardware," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 422–430.

[38] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane." ACM, 4 2017, pp. 164–176.

[39] Â. C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading real-time DDoS attack detection to programmable data planes," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2019, pp. 19–27.

[40] 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, p. 87–95, 2014.

[41] The P4.org Architecture Working Group, "P4 16 portable switch architecture (PSA), version 1.0," 2018. [Online]. Available: https://p4.org/p4-spec/docs/PSA-v1.0.0.html

[42] X. Z. Khooi, L. Csikor, D. M. Divakaran, and M. S. Kang, "DIDA: distributed in-network defense architecture against amplified reflection DDoS attacks," in *6th IEEE Conference on Network Softwarization, NetSoft 2020, Ghent, Belgium, June 29 - July 3, 2020*. IEEE, 2020, pp. 277–281.

[43] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[44] A. Al Sadi, C. Mazzocca, A. Melis, R. Montanari, M. Prandini, and N. Romandini, "P-IOTA: A cloud-based geographically distributed threat alert system that leverages P4 and IOTA," *Sensors*, vol. 23, no. 6, p. 2955, 2023.

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

[46] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric DDoS attacks with programmable switches," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. USENIX Association, 2021, pp. 3829–3846.

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

[48] M. Saquetti, R. Canofre, A. F. Lorenzon, F. D. Rossi, J. R. Azambuja, W. Cordeiro, and M. C. Luizelli, "Toward in-network intelligence: Running distributed artificial neural networks in the data plane," *IEEE Communications Letters*, vol. 25, no. 11, pp. 3551–3555, 2021.

[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] Y. Yuan, O. Alama, J. Fei, J. Nelson, D. R. K. Ports, A. Sapio, M. Canini, and N. S. Kim, "Unlocking the power of inline Floating-Point operations on programmable switches," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. Renton, WA: USENIX Association, Apr. 2022, pp. 683–700.

[51] Intel, "Barefoot tofino." [Online]. Available: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html

[52] "Advanced Programmable Switches (APS), BF2556X-1T Advanced Programmable Switch, APS Networks." [Online]. Available: https://www.aps-networks.com/products/bf2556x-1t

[53] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009, Ottawa, Canada, July 8-10, 2009*. IEEE, 2009, pp. 1–6.

[54] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference, MilCIS 2015, Canberra, Australia, November 10-12, 2015*. IEEE, 2015, pp. 1–6.

[55] G. Engelen, V. Rimmer, and W. Joosen, "Troubleshooting an intrusion detection dataset: the CICIDS2017 case study," in *IEEE Security and Privacy Workshops, SP Workshops 2021, San Francisco, CA, USA, May 27, 2021*. IEEE, 2021, pp. 7–12.

[56] M. Zhang, G. Li, C. Guo, R. Yang, S. Wang, H. Bao, X. Li, M. Xu, T. Wo, and C. Hu, "Superfe: A scalable and flexible feature extractor for ML-based traffic analysis applications," 2025.

[57] D. Koeplinger, M. Feldman, R. Prabhakar, Y. Zhang, S. Hadjis, R. Fiszel, T. Zhao, L. Nardi, A. Pedram, C. Kozyrakis, and K. Olukotun, "Spatial: a language and compiler for application accelerators," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*. ACM, 2018, pp. 296–311.

[58] Y. Zhou, G. Cheng, S. Jiang, and M. Dai, "Building an efficient intrusion detection system based on feature selection and ensemble classifier," *Computer networks*, vol. 174, p. 107247, 2020.

[59] D. Stiawan, M. Y. B. Idris, A. M. Bamhdi, R. Budiarto *et al.*, "Cicids-2017 dataset feature analysis with information gain for anomaly detection," *IEEE Access*, vol. 8, pp. 132 911–132 921, 2020.

[60] J. Gu and S. Lu, "An effective intrusion detection approach using svm with naïve bayes feature embedding," *Computers & Security*, vol. 103, p. 102158, 2021.

[61] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 2016, pp. 4107–4115.

[62] N. Malekghaini, E. Akbari, M. A. Salahuddin, N. Limam, R. Boutaba, B. Mathieu, S. Moteau, and S. Tuffin, "Deep learning for encrypted traffic classification in the face of data drift: An empirical study," *Comput. Networks*, vol. 225, p. 109648, 2023.

[63] G. Andresini, F. Pendlebury, F. Pierazzi, C. Loglisci, A. Appice, and L. Cavallaro, "INSOMNIA: towards concept-drift robustness in network intrusion detection," in *AISec@CCS 2021: Proceedings of the 14th ACM Workshop on Artificial Intelligence and Security, Virtual Event, Republic of Korea, 15 November 2021*. ACM, 2021, pp. 111–122.

[64] D. Roy, P. Panda, and K. Roy, "Tree-CNN: A hierarchical deep convolutional neural network for incremental learning," *Neural Networks*, vol. 121, pp. 148–160, 2020.

[65] R. Bhardwaj, Z. Xia, G. Ananthanarayanan, J. Jiang, Y. Shu, N. Karianakis, K. Hsieh, P. Bahl, and I. Stoica, "Ekya: Continuous learning of video analytics models on edge compute servers," in *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*. USENIX Association, 2022, pp. 119–135.

[66] Q. Zhang, A. Imran, E. Bardhi, T. Swamy, N. Zhang, M. Shahbaz, and K. Olukotun, "Caravan: Practical online learning of In-Network ML models with labeling agents," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 325–345.

# Appendix A. Dataset Feature Description

In this section we provide a full description on the acronym of the features that are present in the three datasets, i.e., NSL-KDD (shown in Table 7), CICIDS-2017 (shown in Table 8) and UNSW-NB15 (shown in Table 9). Specifically, we describe the features shown in the plots of Figure 4.

# Appendix B. Artifact Organization

O'MINE artifact comprises two parts: *(i)* the software simulation with step-by-step instructions for reproducing the principal experiments and figures presented in the paper for two topologies, i.e., linear topology with 5 switches and tree topology with 24 switches; *(ii)* and the FPGA testbed with the necessary scripts to run the FPGA-based evaluation for O'MINE.

TABLE 7: NSL-KDD Feature Description.

| Feature Acronym | Feature Description |
|---|---|
| dst_host_serror_rate | The percentage of connections that have activated SYN errors to the same host |
| dst_host_diff_srv_rate | The percentage of connections to the different services on the same host |
| num_outbound_cmds | The number of outbound commands in an ftp session |
| dst_bytes | The number of bytes sent from destination to source |
| urgent | The number of urgent packets |
| num_root | The number of root accesses |
| dst_host_srv_count | The number of connections to the same service on the same host |
| su_attempted | It is 1 if *su root* command attempted or 0 otherwise |
| rerror_rate | The percentage of connections with *REJ* errors |
| num_shells | The number of shell prompts invoked |
| same_srv_rate | The percentage of connections to the same service |
| is_host_login | It is 1 if the login is a guest login or 0 otherwise |
| logged_in | It is 1 if successfully logged in or 0 otherwise |
| land | It is 1 if the connection is from/to the same host/port or 0 otherwise |
| dst_host_srv_serror_rate | The percentage of connections with SYN errors to same the service on the same host |
| hot | The number of *hot* indicators, e.g., access to system directories |
| srv_diff_host_rate | The percentage of connections to different hosts |
| srv_count | The number of connections to the same service in the past 2 seconds |
| service | The network service on the destination, e.g., http, telnet, ftp |
| dst_host_same_srv_rate | The percentage of connections to same service on the same host |

TABLE 8: CICIDS-2017 Feature Description.

| Feature Acronym | Feature Description |
|---|---|
| Destination Port | The destination port number |
| Bwd Packet Length Min | The minimum length of packets in the backward direction |
| Packet Length Variance | The variance of lengths of packets in the flow |
| URG Flag Count | The number of packets with URG flag set |
| Fwd Packet Length Max | The maximum length of packets in the forward direction |
| Avg Fwd Segment Size | The average size of segments sent in the forward direction |
| Fwd Packet Length Mean | The mean length of packets in the forward directions |
| Fwd IAT Std | The standard deviation of inter-arrival time between packets in the forward direction |
| Avg Bwd Segment Size | The average size of segments sent in the backward direction |
| Flow IAT Std | The standard deviation of inter-arrival time between packets in the flow |
| Fwd Packet Length Std | The standard deviation of packets length in the forward direction |
| Bwd Packet Length Std | The standard deviation of packets length in the backward direction |
| Bwd IAT Total | The sum of all inter-arrival times of packets in the backward direction |
| Packet Length Std | The standard deviation of the length of packets in the flow |
| Bwd Packet Length Max | The maximum length of the packets in the backward direction |
| Bwd Packet Length Mean | The mean length of the packets in the backward direction |
| Max Packet Length | The maximum length of the packets in the flow |
| Init_Win_bytes_forward | The initial window size in bytes in the forward direction |
| Fwd IAT Max | The maximum inter-arrival time between packets in the forward direction |
| PSH Flag Count | The number of packets with PSH flag set |

## B.1. Content

```
O'MINE-Artifact-EuroS&P25
├── software
│   ├── linear
│   ├── pytorch_training
│   ├── reproducibility
│   └── tree
└── hardware
```

- `software/linear` folder contains the scripts to run the experiments in the linear topology that comprises 5 switches;
- `software/pytorch_training` folder contains the scripts for training and testing models;
- `software/reproducibility` folder contains the trained models that run key experiments from the paper and reproduce the corresponding tables and figures;
- `software/tree` folder contains the scripts to run the experiments regarding O'MINE scalability;
- `hardware/` folder contains the scripts and trained models that are used for the O'MINE FPGA-based evaluation.

## B.2. Hosting

O'MINE is hosted on GitHub: https://github.com/bardhienkeleda/O-MINE-Artifact-EuroSP25.git

TABLE 9: UNSW-NB15 Feature Description.

| Feature Acronym | Feature Description |
|---|---|
| dttl | The destination-to-source time-to-live value |
| swin | The source TCP window size |
| dload | The destibation bits per second |
| ct_dst_sport_ltm | The number of connections from the same source port in the last 100 connections |
| ct_dst_src_ltm | The number of connections between the same source and destination IP in the last 100 connections |
| ct_srv_dst | The number of connections to destination IP from the same service |
| sttl | The source-to-destination time-to-live value |
| dtcpb | The destination TCP base sequence number |
| rate | The average number of packets per second sent in a flow |
| ct_state_ttl | The number of flows with the same state and time-to-live value |
| dwin | The destination TCP window size |
| dmean | The mean packet size from destination |
| service | The application layer protocol used, e.g., HTTP, DNS |
| stcpb | The source TCP base sequence number |
| ct_srv_src | The number of connections from a source IP to the same service |
| synack | The time between SYN and ACK |
| ct_dst_ltm | The number of connections to the same destination in the last 100 connections |
| tcprtt | The TCP connection round-trip time |
| sinpkt | The source inter-arrival time |
| ackdat | The time between ACK and data |