Journal of Network and Systems Management (2022) 30:21
https://doi.org/10.1007/s10922-021-09633-5

Machine-Learning-Enabled DDoS Attacks Detection in P4 Programmable Networks

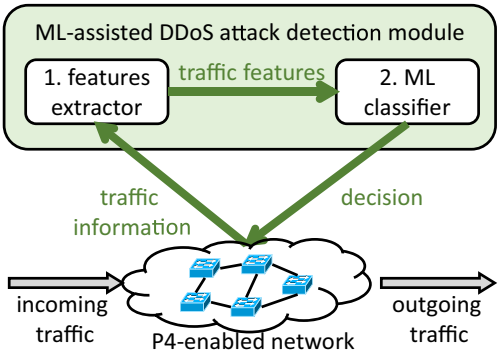
Francesco Musumeci, et al. ***[full author details at the end of the article]***

Received: 21 December 2020 / Revised: 8 September 2021 / Accepted: 29 September 2021 /
Published online: 2 November 2021

Abstract

Distributed Denial of Service (DDoS) attacks represent a major concern in modern Software Defined Networking (SDN), as SDN controllers are sensitive points of failures in the whole SDN architecture. Recently, research on DDoS attacks detection in SDN has focused on investigation of how to leverage data plane programmability, enabled by P4 language, to detect attacks directly in network switches, with marginal involvement of SDN controllers. In order to effectively address cybersecurity management in SDN architectures, we investigate the potential of Artificial Intelligence and Machine Learning (ML) algorithms to perform automated DDoS Attacks Detection (*DAD*), specifically focusing on *Transmission Control Protocol SYN flood attacks*. We compare two different *DAD* architectures, called *Standalone* and *Correlated DAD*, where traffic features collection and attack detection are performed locally at network switches or in a single entity (e.g., in SDN controller), respectively. We combine the capability of ML and P4-enabled data planes to implement real-time *DAD*. Illustrative numerical results show that, for all tested ML algorithms, accuracy, precision, recall and F1-score are above 98% in most cases, and classification time is in the order of few hundreds of μs in the worst case. Considering real-time *DAD* implementation, significant latency reduction is obtained when features are extracted at the data plane by using P4 language.

Graphic Abstract



Keywords Distributed denial of service · Artificial intelligence · Machine learning · Software defined networks · P4 language · TCP flood

1 Introduction

Software Defined Networking (SDN) provides an unprecedented level of network automation with respect to traditional legacy networking [1], mainly due to the functional decoupling between control and data plane and to the logically-centralized network view achieved through dedicated SDN controllers.

However, in SDN, controllers are considered as one of the most critical points of failure and they represent a vulnerable security target. In particular, malicious cyber attacks such as Distributed Denial of Service (DDoS) may affect the controllers in two ways: (i) directly, e.g., when an overwhelming sequence of non-legitimate packets is sent against the controller, impairing its ability to function; and (ii) indirectly, e.g., when attacks against the network nodes result in overflooding the controller with control packets, due to the SDN default forwarding policies configured in the switches. This second case is typical of a stateless SDN approach, where network nodes forward packets based on the flow entries enforced by the controller, while redirecting all unmatched packets to the controller for further instructions.

Recently, SDN has been extended to support stateful data planes, using data plane programmability. In this case, the switch pipeline is programmed to maintain persistent states related to specific network protocols or events (e.g., a layer-4 connection session), thus enabling in-network autonomous per-packet processing, i.e., without the need to interrogate the SDN controller. This way, the SDN switch may be instructed to derive specific statistics and feature extraction from the selected connections.

In this paper we investigate a set of *DDoS attack detection (DAD)* strategies based on Artificial Intelligence/Machine Learning (AI/ML) and leveraging on

SDN stateful data planes, specifically focusing on *Transmission Control Protocol (TCP) SYN flood attacks* [2]. To the best of our knowledge, this is the first time that these two aspects are combined in the context of cybersecurity. The use of stateful data planes is shown to provide a reduction of data forwarding latency and a significantly faster availability of network features needed by the ML algorithms, thus achieving a quicker detection and attack damage minimization. The considered data plane programmability is based on the P4 (Programming Protocol-independent Packet Processors) open source language [3].

This paper extends our previous work [4], where we targeted TCP flood attacks combining the use of ML and P4 to effectively perform DDoS attack detection. Moreover, we propose to combine the ML capability to detect anomaly patterns data with the potential of stateful data planes in processing and collecting traffic information as features, in order to minimize the risk of SDN controller overflooding. To this end, we model the *DAD* as a ML-based classification problem. Using realistic emulated traffic, we compare different ML classifiers and deploy the most suitable algorithm in an "online" scenario where a *DAD* is performed in real time in a ML-based module which directly interacts with the P4-enabled switch. Compared to [4], here we include artificial neural networks among the considered ML algorithms and evaluate algorithms performance not only in terms of classification accuracy, but also in terms of precision, recall and F1-score, which are more suitable metrics especially for unbalanced datasets as in our case. Moreover, we perform a novel comparison, in terms of classification accuracy and prediction time, between two distinct *DAD* architectures, namely, *Standalone* and *Correlated DAD*, where we assume that attacks detection is performed either at the network switches or in a centralized entity (e.g., the SDN controller) exploiting global traffic information, respectively. For the two *DAD* architectures we also evaluate the impact of attack bit rate on the attack detection performance. Furthermore, leveraging P4 language, we evaluate for the two cases the performance of the P4 code combined with the ML classifiers in terms of attack detection time, by comparing three real-time scenarios in which a P4-enabled switch elaborates the received packets in different ways, namely, (1) *packet mirroring*, (2) *header mirroring*, and (3) *P4-metadata extraction*.

The rest of the paper is organized as follows. Section 2 provides a background on DDoS attacks detection and P4 language. In Sect. 3 we describe the ML-assisted *DAD* framework, providing details on the considered *DAD* architectures, ML algorithms and traffic features. Section 4 shows the P4 code adopted for features extraction at the data plane. In Sect. 5 we perform ML algorithms evaluation and model selection, and provide numerical results for the *Standalone* and *Correlated DAD* architectures in Sect. 6. Finally, Sect. 7 concludes the paper.

2 Background

2.1 DDoS Attacks

Denial of Service attacks (DoS) are among the most dangerous cyber security threats affecting server platforms. Such attacks target a server, app or service platforms by sending a huge amount of malicious traffic with the aim to overload its computational (e.g., CPU load level) or network resources (e.g., network interface throughput), thus inducing malfunctioning and/or congestion. The most challenging DoS type is the distributed DoS (DDoS), as a pool of multiple source attackers with different and, often, dynamic/spoofed IP addresses perform a combined attack action. Blocking these types of attacks is difficult since standard IP-address blacklist countermeasures, based on static policies, are not effective. The most utilized DDoS attacks are typically grouped in the following categories: *TCP SYN flood*, *UDP flood*, *ICMP flood* and *HTTP flood*.

Note that TCP/UDP/ICMP packets, generated by any kind of attack type, besides overloading transmission, computing and memory resources in the attack targets (i.e., in the servers), also affect transmission capacity of other network elements (i.e., switches and routers), that need to handle additional traffic generated by the attackers (and also by the victims, when responding to the attack packets).

Among these attack types, in this paper we focus on *SYN flood attacks* [2], which represent one of the most relevant DDoS attacks.

TCP SYN flood attacks exploit TCP connections' initiation packets to target the victim. Usually, the attacker sends multiple SYN requests from multiple spoofed IP addresses, but does not reply to the victim SYN-ACK packet. This way, memory and computing resources at the victim's system are unnecessarily allocated while waiting for the ACK messages required to successfully terminate the TCP connections handshake from all the senders.

According to [5], three defense strategies are typically employed to mitigate DDoS attacks, classified based on the location of the detection engine:

- *Source-based* detection, implemented at the attacking hosts
- *Destination-based* detection, implemented at the victim hosts
- *Network-based* detection, implemented at the network intermediate nodes (e.g., switches, routers)

The objective of this paper is to perform *in-network* attacks detection by deploying defense mechanisms directly at the SDN switches with the aim of blocking the attack at the data plane level and preserving the SDN controller from major malfunctioning or out-of-service events. Therefore, we focus on a network-based defense mechanism.

2.2 Related Work

A number of ML-based detection strategies for DDoS attacks have been proposed in literature, relying on ML ability to automatically infer anomaly patterns in a sequence of traffic packets [5]. Several studies proposed to use Support Vector Machine (SVM) classifiers, such as [6–10], and [11]. artificial neural networks, such as [12, 13] and [14], and other ML algorithms, as in [15–19], and [20]. Interestingly, reinforcement learning has been also adopted to perform DDoS mitigation in paper [21].

Other papers have addressed attacks detection in the specific context of SDN, such as [22–33], and [34].

Concerning the recent trend of considering SDN stateful data planes for DDoS attack mitigation, a significant amount of work has appeared recently, though not fully exploiting sophisticated detection mechanisms such as those based on ML algorithms. For example, authors of [35] proposed a distributed architecture of stateful switches to mitigate attacks as an alternative to classical SDN centralized solutions, which are potentially more prone to computational resource bottlenecks. Moreover, authors of [36] presented an alternative model for the coordination of stateful switches. In [37], authors leveraged P4 to enable traffic inspection for real-time attack detection, whereas authors of [38] adopt P4 language and statistical models based on IP address entropy to distinguish between legitimate and attack traffic. Similarly, in [39], authors implemented a P4 strategy to contrast TCP flood port scan attacks and evaluated this strategy in both a P4-enabled software switch and a FPGA. Authors in [40] and performed attack detection in P4-programmable Ethernet switches, focusing on SIP attacks. Furthermore, authors of [41], data plane programmability is exploited to mitigate DDoS attacks of different types, such as SYN flood, DNS amplification, HTTP flood, when traffic characteristics change over time, by adopting threshold-based defense mechanisms. Finally, authors of [42] used P4 programming to contrast diverse attacks also taking into account the QoS of legitimate users.

2.3 P4 Language

As mentioned before, in SDN, a DDoS attack targeting the SDN controllers may seriously affect not only the correct functioning of the controllers, but also the overall network stability and operation, due to the logically centralized nature of the SDN control plane. For such reason, it is crucial to keep the controller involvement in the *DAD* process as limited as possible, as a large number of computationally-intensive packet inspections may affect controller stability. In this context, programmability of SDN data plane offers a new opportunity to perform traffic inspection inside the switches at wire-speed.

In this paper we exploit the P4 (Programming Protocol-independent Packet Processors) open source language [3] to program a SDN switch pipeline with the ability to perform traffic feature extractions to be consumed by ML algorithms. P4 is a high-level, vendor-independent language and has been designed to enable

custom-programmed pipelines and forwarding planes on SDN switches, not constrained by traditional fixed-functions protocol stack. The compiled P4 code is submitted to a programmable device backend (i.e., a programmable network interface card, a bare metal switch, a FPGA, a software switch) in charge of enforcing the desired pipeline structures and functions.

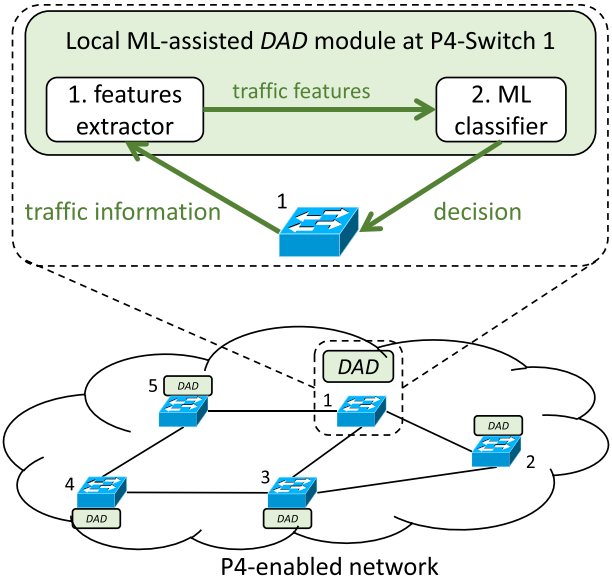
A typical P4 code is structured by some well-defined components:

- *Parsers*, responsible for the analysis of the incoming packet and the detection of the considered protocol stacks (either standard or proprietary)
- *Tables and actions*, the key set of the SDN paradigm, identifying the packet processing rule in the standard match-action fashion. In P4, tables and actions may be programmed with a high level of flexibility and rule definitions, including protocol field updates, port selection, actions on the entire packet (e.g., packet drop, cloning, recirculation)
- *Pipeline control*, responsible of structuring a programmable set of tables inside a given pipeline, in the context of well-defined pipeline abstract models. In all the considered models, the design identifies an ingress pipeline (i.e., performing operations at packet reception and implementing forwarding decisions) and an egress pipeline (i.e., operations performed after the forwarding decision, such as pre-forwarding operations or multicast). Each pipeline defines an ordered sequence of tables, optionally subject to conditional rules and loop execution. This latter feature is a specific and powerful P4 feature with respect to traditional SDN pipelines, typically implemented with a static set of tables.

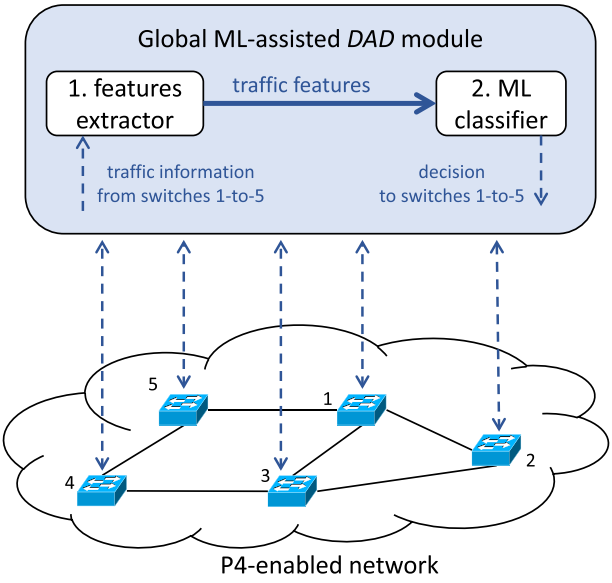
In addition, P4 is able to define and manipulate packet metadata, utilized to associate extra information to the packet and performing further processing. Examples of typical metadata are timestamps, features, states, processing latency, etc. Finally, P4 allows to define and allocate stateful objects (i.e., memory persistent variables inside the switch) that may be used to activate context-based processing and implement Finite State Machines. In particular, P4 defines the use of meters (i.e., a three-state object used to measure and classify the throughput of a given flow), counters and registers. Such objects can be read/write accessed and utilized to perform stateful-based actions inside tables.

3 ML-Assisted DDoS Attack Detection

The aim of this paper is to perform *DAD* in SDN networks by combining ML ability in performing effective attack detection by automatically retrieving traffic information (i.e., a *signature*), and the opportunity to perform packet processing directly at the data plane, enabled by stateful data planes and P4 language. Moreover, we compare two *DAD* architectures, namely, *Standalone* and *Correlated DAD*, in terms of classification performance and algorithm complexity, i.e., training duration and prediction time, also evaluating the impact of attack rates on algorithms performance.



(a) *Standalone DAD*



(b) *Correlated DAD*

Fig. 1 *Standalone* and *Correlated* DDoS Attack Detection Architectures

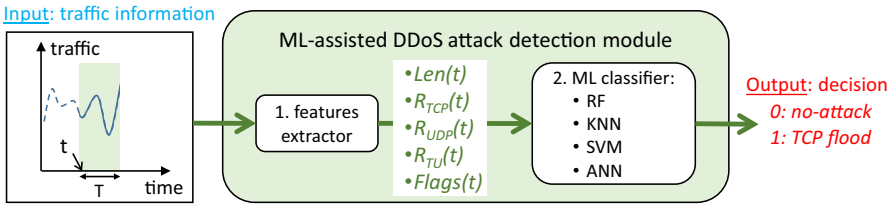


Fig. 2 Window features extraction and classification

3.1 *DAD* Detection Architectures

We define two different DDoS attacks detection architectures, namely, *Standalone DAD* and *Correlated DAD*, whose functional blocks are illustrated in Fig. 1a, b, respectively, for a network with 5 P4-enabled switches. For both *Standalone* and *Correlated* architectures, we model the DDoS attack detection into a ML classification problem. Given the traffic (i.e., a series of packets arriving at one or more P4 switches) observed in a certain time frame, i.e., a *time window* with pre-defined duration T, the detection module outputs a decision for the observed traffic, i.e., a label "*1: attack*" or "*0: no-attack*", to indicate if in the considered time frame an attack is present or not, respectively. This label allows decision-making on packet forwarding, which may consists of, e.g., dropping packets after a number of subsequent windows are classified as containing an attack, or even performing further analysis by, e.g., forwarding selected packets to the SDN controller. Note that, in this paper, we do not concentrate on specific packet forwarding decisions, but limit our analysis on the binary classification of traffic windows of duration T.

As shown in Fig. 1a, in the *Standalone DAD* architecture, a ML-assisted *DAD* module is deployed at each P4 switch, so that each switch performs DDoS attacks detection only based on locally-observed traffic. Conversely, in the *Correlated DAD* architecture (see Fig. 1b), a unique *DAD* module receives traffic information from several P4 switches and takes decisions based on globally-observed traffic.[1]

The detection module is generally constituted by two operational blocks, i.e., (1) *features extractor* and (2) *ML classifier*. However, in both *Standalone* and *Correlated* architectures, the detection module can be simplified by offloading some operations (e.g., features extraction or even the ML-based classification) directly at the data plane in the P4 switches. In such a scenario, we also evaluate the additional latency introduced by the attack detection module considering that information derived from traffic flows is exchanged between the detection module and the P4 switches in different forms, e.g., by mirroring entire data packets, their headers, or even extracting metadata (i.e., features) from a sequence of packets.

[1] Note that, in the *Correlated* architecture, the *DAD* module can be co-located with the SDN controller, or even physically deployed at one of the P4 switches, receiving traffic information from other P4 switches using out-of-band channels. Analysis considering specific deployments of the *DAD* module are out of the paper scope, so we assume that the *DAD* module is deployed out of all the switches in the network, without affecting the generality of our analysis.

3.2 ML Classifiers and Considered Features

To implement the classifiers for *DAD* we consider four different ML algorithms, namely, Random Forest (RF), K-Nearest Neighbours (KNN), Support Vector Machine (SVM) and Artificial Neural Network (ANN). A comparison between the various algorithms has been carried out to devise the most appropriate solution in terms of classification performance and algorithm complexity, i.e., training duration and prediction time.

We developed binary classifiers that, for each window of duration T, assign one of the following two labels: *"0: no-attack"* or *"1: TCP flood"*. A summary of the steps performed by the ML-assisted *DAD* module is shown in Fig. 2, along with the list of features extracted from the time window. Specifically, for each traffic window of duration T starting at a generic time instant t, we consider the following features f_1 to f_5:

– *Average length*: the average size in bytes of packets in time window $(t, t+T)$

$$f_1 = Len(t) = \frac{total\ no.\ of\ bytes\ in\ (t, t+T)}{total\ no.\ of\ packets\ in\ (t, t+T)}$$

– *TCP ratio*: the percentage of TCP packets out of the total in time window $(t, t+T)$;

$$f_2 = R_{TCP}(t) = \frac{no.\ of\ TCP\ packets\ in\ (t, t+T)}{total\ no.\ of\ packets\ in\ (t, t+T)}$$

– *UDP ratio*: similarly to R_{TCP}, it represents the percentage of UDP packets;

$$f_3 = R_{UDP}(t) = \frac{no.\ of\ UDP\ packets\ in\ (t, t+T)}{total\ no.\ of\ packets\ in\ (t, t+T)}$$

– *TCP-UDP ratio*: the ratio between TCP and UDP packets in time window $(t, t+T)$. If no UDP packet is present in the window, we set this feature equal to a large finite number;

$$f_4 = R_{TU}(t) = \frac{no.\ of\ TCP\ packets\ in\ (t, t+T)}{no.\ of\ UDP\ packets\ in\ (t, t+T)}$$

– *Flags(t)*: is the percentage of TCP packets with an active SYN flag out of the total in time window $(t, t+T)$.

$$f_5 = Flags(t) = \frac{no.\ of\ TCP\ packets\ with\ SYN\ flag\ in\ (t, t+T)}{total\ no.\ of\ packets\ in\ (t, t+T)}$$

Note that, although several traffic features have been adopted in literature to perform *DAD* [43], as we focus on TCP flood attacks, we selected features according to the considered attack type and following traffic information typically used in literature [43–45]. Moreover, as in our paper we aim at performing attack detection independently from the attacker or victim location, among the features considered in the

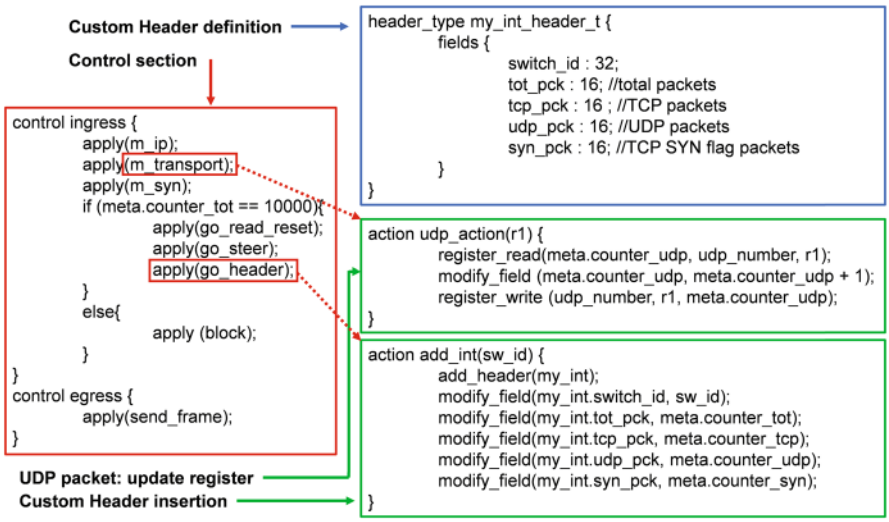


Fig. 3 Selected P4-metadata extraction code sections: ingress pipeline control, extra header and key actions

aforementioned papers, we ignore location-specific ones, such as IP source/destination addresses and TCP/UDP ports.

4 Using P4 Language for Attack Detection in Data Plane

The aforementioned ML classifiers need to acquire real-time traffic data to perform attack detection. The way real-time data are elaborated to perform features extraction and feed ML classifiers represents a key aspect in the efficiency of the whole detection system. In the case of raw traffic data (e.g., traffic mirroring), features need to be extracted by the *DAD* module through deep packet inspection techniques. The availability of a P4 programmable data plane allows to offload the feature extraction, enabling the deployment of selected P4 switches able to derive the traffic features at wire-speed for immediate submission to the ML classifiers, thus speeding up the detection process. For this reason we have implemented three versions of feature extraction to be implemented at P4 switches: a) packet mirroring, b) header mirroring, c) metadata extraction.

Packet mirroring is the simplest version, in which packets to be analyzed are directed to the *DAD* module for feature extraction and classification. Header mirroring represents an intermediate version, where mirrored packets are truncated to preserve the protocols header stack, mainly in order to reduce the throughput of the data subject to feature extraction and, ultimately, the processing burden at the *DAD* module. Metadata extraction requires the most complex P4 implementation, as features are extracted directly at the P4 switch exploiting telemetry functions. Differently from in-band telemetry[46], where metadata are sent in-band

and exchanged/elaborated by the SDN domain switches, here feature extraction is configured as out-of-band telemetry data collected by each switch and sent to control/monitoring interface to be consumed by the *DAD* module. Three specific P4 language aspects have been exploited to realize this goal:

- *Stateful objects handling*, with the definition of programmable registers storing and updating the number of selected packets occurrences within a traffic window;
- *Feature Extra header*, used to convey the statistics and provide the analysis results to the detection module utilizing a portion of selected mirrored packets.
- *Conditional pipeline control*, used to implement different pipeline execution branches subject to context condition.

The excerpts of Fig. 3 show a portion of selected sections of the P4 code used to deploy extract metadata in P4 switches. First, the code defines all the required protocol stack headers and the related parsers to perform packet inspections: Ethernet, IP, UDP and TCP headers. Then, four registers are defined to store the number of IP, UDP, TCP and TCP SYN packet occurrences within a given traffic window. In addition, the code defines a proprietary extra header, namely *my_int_header_t* utilized for the report packet to the *DAD* module, as depicted in the figure. The header is composed by a 4-byte long *switch_id* field used to identify the source switch address sending the report, along with four 2-byte long fields used to convey the number of IP, UDP, TCP and TCP SYN packets observed in the traffic window, i.e., the features utilized by the *DAD*. Finally, a custom-defined packet metadata (i.e., *meta*) is defined to associate to the analyzed packet extra information, such as the cumulative packet number inside the traffic window (i.e., *meta.counter_tot*).

In the following the workflow of the P4 code is explained. First, the received packets are parsed to extract the considered protocol headers. Then, the code enters the ingress pipeline, defined by the control block shown in the figure. The ingress pipeline includes the sequence of three tables, used to perform the statistics. In particular, each table (i.e., `m_ip` for IP, `m_transport` for UDP/TCP and `m_syn` for SYN flag detection) performs the update of the feature occurrences in the registers and in the packet metadata. The figure shows the list of actions related to the `m_transport` table when the match detects UDP packets: in this case the register at offset `r1` will be first read (i.e., to retrieve the last cumulative value of UDP packets occurrences), incremented and re-written by means of an auxiliary packet metadata field. Similarly, such operations are performed in the other two tables. When the statistics have been updated, the ingress pipeline is subject to a conditional check. In the case the traffic window (e.g., set to 10^5 packets in the P4 code, see experimental results) is terminated (the check is done using the specific packet metadata field), the code executes a specific branch of tables (`go_read_reset`, `go_steer` and `go_header` in Fig. 3) in order to generate the report packet to the *DAD*, otherwise it follows a standard packet forward/block procedure based on flow entries received by the *DAD* module or by the SDN controller. In this specific implementation, the traffic window is mapped in a packet-based window assuming that

the switch operates in a constant bit rate scenario (as in the experimental evaluation, see Sect. 6.3), while in a more general deployment it may be mapped in a time-based window using P4 timestamp metadata. Forwarding is implemented using standard network-layer information to emulate internet router behavior. Specifically, in the case of packet report generation, table `go_read_reset` is responsible for resetting the values of the internal registers, table `go_steer` is responsible for cloning the packet and set its output port to the control interface connected to the *DAD* module, while table `go_header` triggers the features extra header insertion, positioned after the considered packet protocol stack. The figure shows the details of the action `add_int`, inside table `go_header`. The action first generates the extra header insertion in the packet (i.e., `add_header` P4 native command), then updates its fields with the statistics retrieved by the registers and temporarily stored in the packet metadata, along with the switch id parameter, thus allowing multiple network switches to send their reports to the *DAD* in parallel. It is worthwhile to note that a P4 switch is not allowed to generate a new asynchronous packet, thus the report packet sent to the *DAD* is the result of a mirror and a subsequent manipulation (i.e., extra header inclusion) of an existing traffic packet. The final P4 behavior allows to generate a features report at the end of each traffic window, ready to be submitted to the classifier. In the meanwhile, the switch is able to act as a firewall allowing/blocking suspected flows indicated by the controller through the *DAD* module outcomes.

It is important to underline that the selection of the features is a key aspect of the programmable data plane effectiveness and scalability. In fact, the considered stateful features are generated, processed and stored inside the P4 switch resorting to stateless transport-layer information retrieved by packet parsing stage after flow match condition. Such strategy allows overall system scalability, since P4 switches have been demonstrated to scale with the number of flow entries (e.g., number of different flows analyzed using the same pipeline control section) [39]. Conversely, the online stateful analysis of TCP sessions would require a relevant processing burden, practically unfeasible in metro and core routers due to the high amount of TCP connections and with noticeable scalability issues, even for a P4 switch.

5 Case Study and ML Algorithms Settings

5.1 Traffic Scenario and Corresponding Datasets

ML algorithms have been implemented with Python-based scripts using *keras* and *sklearn* libraries on a desktop with 8×2 GHz processor and 8 GB of RAM. Traffic data for training and testing of our algorithms has been collected using a Spirent N4U traffic generator [47]. We generate realistic traffic traces for 15 min, where TCP SYN flood attack traffic at an average bit rate of 26.5 kbit/s is added to regular background traffic at 30 Mbit/s. The attack traffic is designed as an aggregation of flows having random source IP addresses and specific IP destination addresses, according to the nature of DDoS attacks. Moreover, the sequences utilize incremental TCP port scanning with random initial values and duration. The low rate has been selected with the aim to test the system detection sensitivity. Average duration

Table 1 Traffic parameters in the datasets

Parameter	Value
Traces duration	15 min
TCP traffic bit rate	13.5 Mbit/s
UDP traffic bit rate	11.4 Mbit/s
IP traffic bit rate	5.1 Mbit/s
Attack traffic bit rate	26.5 kbit/s
Attack Type	SYN flood
Window duration	$T \in \{0.5;1;2;10\}$ s
Windows distance	$\delta \in \{0.01;0.05;0.1;0.2;0.5;1\}$ s

of the attacks is 10 s, whereas background traffic is composed by three different flows, i.e., 13.5 Mbit/s TCP traffic, 11.4 Mbit/s UDP traffic and 5.1 Mbit/s IP traffic (not carrying UDP/TCP payloads[2]), with packet length following Internet Mix (IMIX[3]) distribution [47].

Starting from this traffic trace, we create different datasets extracting traffic windows of duration T and collected at a sampling period δ. For our analysis, we consider different values for parameters T and δ and, for each of the datasets obtained varying the values of T and δ, we label the windows by assigning label *"1: TCP flood"* only when the window contains at least one packet belonging to the TCP flood attack, otherwise the window is assigned label *"0: no-attack"*. Note that, varying the value of parameter δ (i.e., the distance between two consecutive windows), the total number of windows in the dataset varies accordingly, ranging between 1800 and 180000 windows, for the cases of $\delta = 1$s and

[2] The considered background traffic profile in terms of protocols has been designed according to the expected Internet traffic profiles at Tier 1/2 carrier router in the next years. Instead of considering the historical protocol distribution (where TCP dominates with over 70% volume traffic [48]), we designed the profile according to the two most significant recent trends: the rapid increase of the HTTP/3 QUIC protocol running on UDP, supported by Google, Facebook and other major platforms, that will replace HTTP/2 (running on TCP), and the increase of IPv6 traffic, towards 20% volume rate [48]. Google network volume traffic is dominated by QUIC, more than 40% [49, 50]. Current overall QUIC rate (around 10%) is expected to converge rapidly to the Google value once big providers will migrate their web protocols to HTTP/3. For these reasons we modelled 38% UDP traffic and 17% IP raw traffic, while the dominant part is still TCP (45%). We remark that, given the features adopted in the ML algorithms, which are mainly based on packets length and proportions of TCP/UDP packets and SYN flags out of the total number of packets in a certain time frame (see Sect. 3.2), the absolute values of bit rates used in the considered traffic scenario do not affect relevantly the numerical results due to the fact that we perform features scaling and normalization, therefore the values of features $f_1 \div f_5$ would not be affected if all traffic flows were increased by a given common factor.

[3] IMIX traffic distribution is based on statistical sampling done on Internet routers. Traffic profile definitions are based on IETF RFC 6985 [51] and the tests are run in accordance with RFC 2544 [52]. The considered profile, defined as a Table of Proportions [51], defines the following distribution: 60-byte (58.33%), 576-byte (33.33%), 1500-byte (8.33%). The profile was motivated by the fact that it is the most considered traffic profile for internet routers used for tests and measurements, that guarantees reproducible traces.

$\delta = 0.01$s, respectively. The parameters considered in our analysis are summarized in Table 1.

5.2 Evaluation Metrics

In this study we compare different ML-based *DAD* solutions in terms of classification performance and complexity. More specifically, since we model attack detection as a binary classification, where we distinguish among *"positive"* and *"negative"* windows (respectively, windows with label *"1: TCP flood"* or *"0: no-attack"*), we consider:

- *true positives (TP)*: Number of windows of type '1' correctly classified with label '1';
- *true negatives (TN)*: Number of windows of type '0' correctly classified with label '0';
- *false positives (FP)*: Number of windows of type '0' misclassified with label '1';
- *false negatives (FN)*: Number of windows of type '1' misclassified with label '0'.

Based on these definitions, to evaluate classification performance, we use the following metrics:

- *Accuracy* is the fraction of correctly-classified windows, i.e.,

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

- *Precision* is the fraction of correctly-classified positive windows out of the total number of windows classified as positive, i.e.,

$$P = \frac{TP}{TP + FP}$$

- *Recall* is the fraction of correctly-classified positive windows out of the total number of windows which are actually positive, i.e.,

$$R = \frac{TP}{TP + FN}$$

  Note that *precision* and *recall* are two contrasting objectives and different algorithms may provide different trade-offs on these measures.
- *F1-score* (or *F-score*) is used as a unique metric when both precision and recall are relevant in the evaluation, and it is defined as follows:

$$F1 = 2\frac{P \cdot R}{P + R}$$

Concerning algorithm complexity, we evaluate the four ML algorithms considering the following metrics:

Table 2 Hyperparameters selection for the various ML algorithms

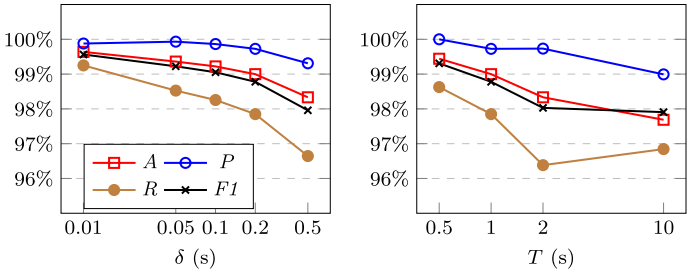
Algorithm	Parameter	Tested values	Selected value
KNN	no. of neighbors K	{3, 4, 5, 6, 7, 8, 9, 10}	3
	neighbors weight	{uniform, distance-based}	uniform
RF	splitting criterium	{Gini, Entropy}	Gini
	no. of trees	{10, 20, 30, 40, 50, 60, 70, 80, 90, 100}	10
SVM	kernel	{sigmoid, rbf, polynomial}	rbf
	regul. param. C	$\{1, 10, 10^2, 10^3, 10^4\}$	10^3
	kernel coefficient γ	$\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1\}$	10^{-2}
ANN	no. of hidden layers	{1,2}	2
	no. of neurons per layer	{5,10,11}	10
	activation function	{sigmoid, relu, elu}	elu

– *training duration*: it represents the time required to perform ML algorithm training; as in the following we adopt fivefold cross-validation to perform algorithms evaluation, we show training duration as an averaged value across all the subsets used for algorithm training, i.e., it is evaluated on 1/5 of the entire dataset obtained with given values of T and δ.
– *test time*: it is the time needed to perform classification of a single traffic window once the ML algorithm has been trained; note that, for each ML algorithm, test time value can vary with window duration T, but it is not affected by window sampling period δ.

5.3 ML Models Selection

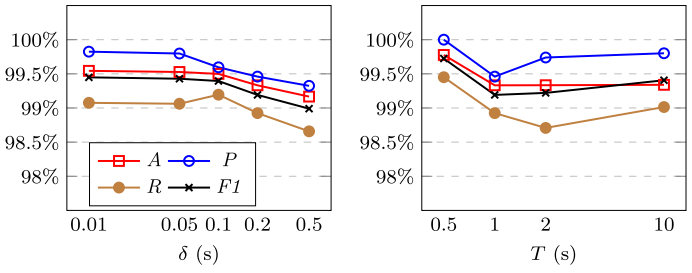
We consider four ML algorithms to perform windows classification for DDoS attacks detection, namely, RF, KNN, SVM, and ANN. For each algorithm, different combinations of hyperparameters have been evaluated using fivefold cross-validation, in order to obtain the classifiers with high classification accuracy (i.e., above 97%) and sufficiently-low training duration. For this analysis, we consider fixed values of window duration and window sampling interval (i.e., $T = 1$ s and $\delta = 0.2$ s, respectively), in order not to affect ML model selection. These values have been selected as they were obtained as best performing values for the real-time implementation of *DAD* in [4], however a further sensitivity analysis on the values of T and δ will be shown in the following after ML model selection, i.e., when the hyperparameters of all ML algorithms have been decided.

For each ML algorithm, the combinations of hyperparameters which have been evaluated (such as number of hidden layers and hidden neurons in ANNs, kernel in SVM, number K of neighbors in KNN, splitting criteria in RF, etc. [53]) are reported in Table 2, along with the selected hyperparameters.



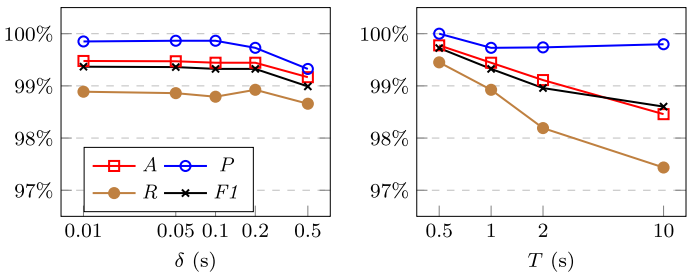
(a) Comparison over δ ($T = 1$ s) **(b)** Comparison over T ($\delta = 0.2$ s)

Fig. 4 Classification performance for KNN algorithm, varying parameters (a) δ and (b) T (A = *Accuracy*, P = *Precision*, R = *Recall*, $F1$ = $F1$-*score*)



(a) Comparison over δ ($T = 1$ s) **(b)** Comparison over T ($\delta = 0.2$ s)

Fig. 5 Classification performance for RF algorithm, varying parameters (a) δ and (b) T (A = *Accuracy*, P = *Precision*, R = *Recall*, $F1$ = $F1$-*score*)



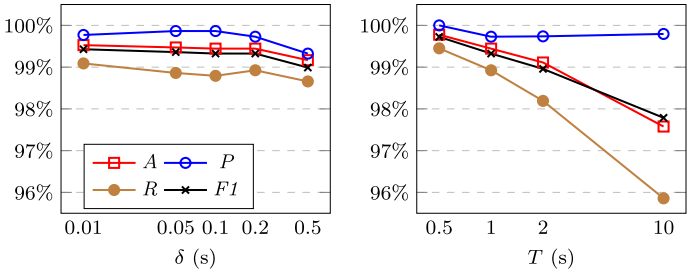
(a) Comparison over δ ($T = 1$ s) **(b)** Comparison over T ($\delta = 0.2$ s)

Fig. 6 Classification performance for SVM algorithm, varying parameters (a) δ and (b) T (A = *Accuracy*, P = *Precision*, R = *Recall*, $F1$ = $F1$-*score*)

6 Numerical Results

6.1 ML Algorithms Performance Evaluation

We start the numerical analysis of ML-assisted *DAD* by evaluating the impact



(a) Comparison over δ ($T = 1$ s) **(b)** Comparison over T ($\delta = 0.2$ s)

Fig. 7 Classification performance for ANN algorithm, varying parameters **a** δ and **b** T ($A = Accuracy$, $P = Precision$, $R = Recall$, $F1 = F1\text{-}score$)

Table 3 Mean training time for the different ML algorithms and varying δ ($T = 1\,s$)

Algorithm	$\delta = 0.01$ s	$\delta = 0.05$ s	$\delta = 0.1$ s	$\delta = 0.2$ s	$\delta = 0.5$ s
KNN	0.153	0.03	0.015	0.009	0.009
RF	1.003	0.236	0.154	0.126	0.128
SVM	3.28	0.179	0.077	0.03	0.014
ANN	10.303	6.784	5.289	5.239	5.278

of parameters T and δ on ML algorithms performance (considering the metrics described in Sect. 5.2) and focusing on the *Standalone DAD* architecture depicted in Fig. 1a. Values considered for parameters T and δ are shown in Table 1.[4] For each case, since we use fivefold cross-validation, the numerical results shown in the following, are averaged across all the five folds, except for test time, which is measured as the classification time for a single window of duration T. Concerning ML algorithms hyperparameters, we consider only the values obtained after ML model selection, which are reported in the right-most column of Table 2.

We first concentrate on Accuracy (A), Precision (P), Recall (R), and F1-score ($F1$), which are shown in Figs. 4, 5, 6, and 7 for KNN, RF, SVM and ANN algorithms, respectively, and for increasing values of δ and T (respectively, subfigures (a) and (b)). When one of the two parameters is varied, the other one is kept at a fixed value (respectively, $T = 1$ s and $\delta = 0.2$ s), on the line of the analysis done in Sect. 5.3.

As expected, in general, increasing the value of δ, provides performance deterioration for all the metrics and independently from the ML algorithm under analysis, caused by the decrease of data points in the dataset (hence, a lower number of data points used for training) when increasing δ. For all values of δ, extremely-high performance is obtained, with all metrics laying above 96.6% for KNN and 98.6% for RF, SVM and ANN algorithms, respectively. Notably, the values of precision P are above 99% for all algorithms, showing that the classification of positive examples

[4] Recall that, for each T-δ pair, a different dataset is obtained.

Table 4 Average test time for the different ML algorithms ($\delta = 0.2$ s, $T = 1$ s)

Algorithm	Mean test time (μs)
KNN	113.18
RF	112.68
SVM	1.75
ANN	279.95

(i.e., windows containing at least one attack packet) performed with any of the ML algorithm is highly reliable. On the other hand, the values of recall *R* are the lowest ones among all performance metrics and for all the four algorithms, meaning that, despite the good performance of all the algorithms, still a very small percentage of windows affected by attacks are misclassified as legitimate. This aspect suggests that, in realistic deployments, further analysis, e.g., performed at the *DAD* module (possibly co-located with the SDN controllers), may be necessary on some of the windows classified as attack-free by the ML-based *DAD*.

Concerning the performance of the various algorithms when varying window duration *T*, it is observed that increasing *T* deteriorates in general all performance metrics, since the relatively-low amount of attack packets in longer windows do not allow to efficiently capture the attack characteristics through the considered features. Only in the KNN and especially RF cases, when *T* is increased above a certain value (i.e., above 2s and 1s, respectively) the performance metrics start increasing after an initial decrease. Similarly to the variation of δ, also when varying *T* the performance deterioration is mainly observed for the recall *R*, confirming that, depending on the values of *T*, the role of *DAD* module may still be crucial to perform further analysis onto windows classified as negatives by the ML algorithm. Therefore, it is evident that, besides a sufficiently large dataset (i.e., a sufficiently low value of δ), fine-tuning of window duration *T* is also necessary to avoid overloading the *DAD* module.

To assess their complexity, we now compare the four ML algorithms in terms of training and test time. Table 3 shows the mean training time[5] for the four ML algorithms and for increasing values of δ. We here consider a fixed value of $T = 1$ s as we observed that window duration *T* does not affect training and test time significantly (we do not report such an analysis over *T* due to space limitations). As expected, training time decreases when increasing δ, due to the lower number of data points used for training. ANN shows the worst training time, which is up to 500 times higher than all other algorithms, especially for higher values of δ. Indeed, it can be observed that dataset size has a significant impact on RF and especially SVM, for which the training time is reduced from 3.28 *s* (for $\delta = 0.01$ s) to 0.01 *s* (for $\delta = 0.5$ s). Finally, KNN training time is negligible for all values of δ, sincet KNN is a non-parametric ML algorithm, and so no real training phase is necessary, but only hyperparameter selection is performed.

[5] Note that, as we adopt fivefold cross-validation, training time is expressed as the an averaged value across all the subsets used for algorithm training, i.e., it is evaluated on 1/5 of the entire dataset obtained with given values of *T* and δ.

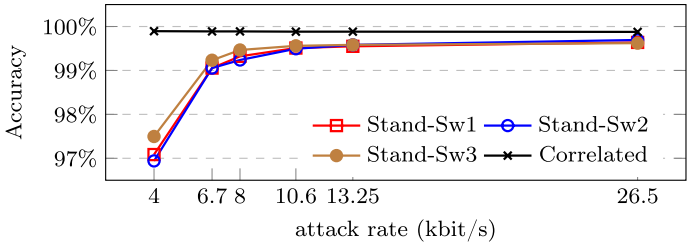
To evaluate classification speed in a possible real-time implementation of *DAD*, we compare the four ML algorithms in terms of test time, and show in Table 4 the values obtained for the classification of a single data point after model training has been performed, considering $T = 1$ s and $\delta = 0.1$ s. Results are shown in terms of *mean* test time, i.e., for all the four ML algorithms, we average classification time over all data points in the test set. Observing the results, SVM shows the smallest test time, which is two order of magnitude lower than the test time for the other algorithms. On the other hand, ANN shows the worst performance, with test time which is doubled in comparison to KNN and RF.

6.2 *Standalone* and *Correlated DAD* Architectures

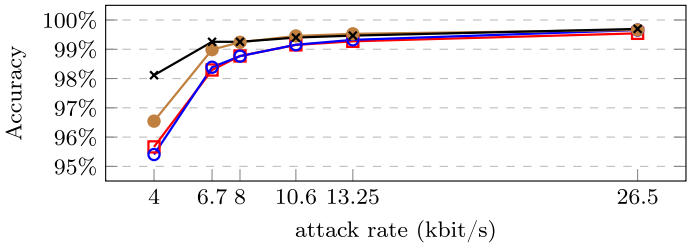
In this subsection we compare the *Standalone* and *Correlated DAD* architectures discussed in Sect. 3.1, considering a sample network with 3 P4-enabled switches. To perform our analysis, we start from the same traffic traces with characteristics discussed in Sect. 5.1 and tailor two distinct datasets for the *Standalone* and *Correlated* scenarios. In particular, for the *Standalone* case, we randomly split legitimate and attack packets into three equally-sized subsets, one for each of the three switches, and form windows of duration T taken at distance δ to build the three datasets. To have a homogeneous comparison between the *Standalone* and *Correlated* scenarios, in both cases windows of duration T are labeled as 'positive' (i.e., label = 1) if at least one attack packet is included in the window of *any* of the three switches. We remark also that, while in the *Standalone DAD* architecture each switch operates window classification independently from the other switches (i.e., based on the *local* windows features f_1 to f_5 discussed in Sect. 3.2), in the *Correlated DAD* architectures, classification of a given window of duration T is performed based on the overall set of 15 features collected from all the three switches in time-frame T.

To evaluate the performance of the *Standalone DAD* architecture, we compare it with the *Correlated DAD* in terms of classification accuracy, considering different amounts of attack packets out of the total observed traffic. To do so, for both *Standalone* and *Correlated* cases, we generate 6 different scenarios considering attack traffic bit rate at 26.5, 13.25, 10.6, 8, 6.7 and 4 kbit/s, corresponding to the case of 100%, 50%, 40%, 30%, 25% and 15% of the maximum attack rate considered so far, respectively.

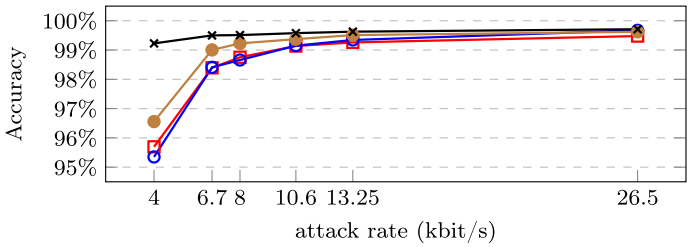
As expected (see Fig. 8), in the *Correlated* architecture classification accuracy is higher compared to the accuracy obtained at any switch in the *Standalone* case, independently from the adopted ML algorithm. This is due to the global traffic information that can be exploited in the *Correlated* scenario, which is more significant for lower attack rates, whereas when attack traffic becomes more relevant, i.e., above 8 kbit/s for KNN and above 13.25 kbit/s for the other algorithms, accuracy of *Standalone DAD* is always above 99% and approaches the performance of *Correlated DAD*. We remark that, although low rate SYN floods might not be extremely dangerous in traditional network environments, in SDN scenarios they might increase the probability of service degradation, due to the fact that many switches/routes can



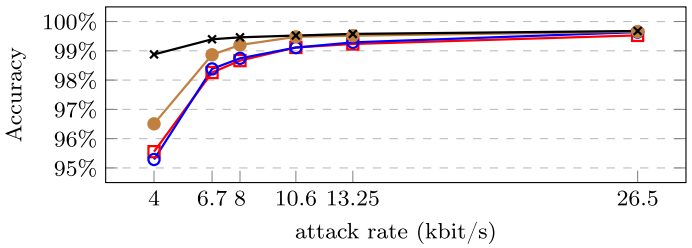
(a) KNN



(b) RF



(c) SVM



(d) ANN

Fig. 8 Classification accuracy for *Standalone* and *Correlated DAD* architectures for different ML algorithms and increasing attack rates ($T = 1$ s, $\delta = 0.01$ s)

redirect to the same SDN controller several packets which do not match any entry of their flow tables.

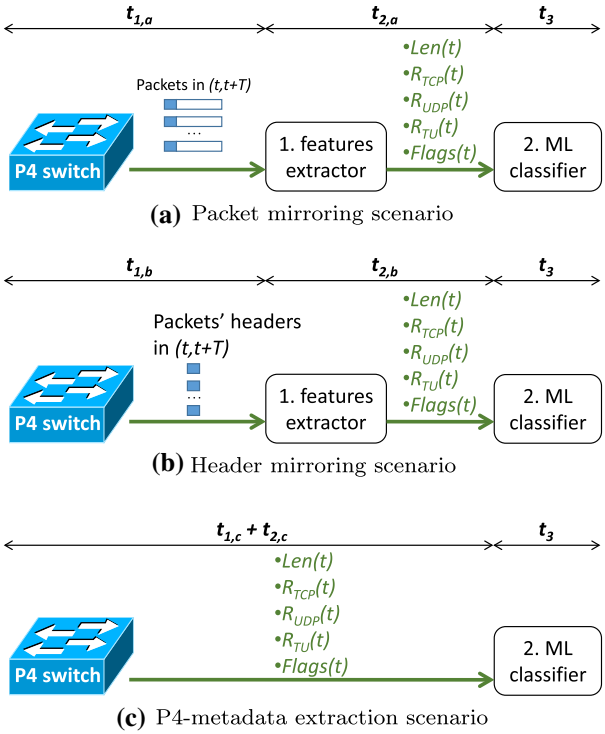


Fig. 9 Different scenarios and corresponding time contributions

Table 5 Time contributions for different scenarios and for the *Standalone DAD* architecture (t_3 values in parenthesis represent classification time for the *Correlated DAD* architecture)

	RF			SVM		
	t_1(µs)	t_2 (s)	t_3(µs)	t_1(µs)	t_2 (s)	t_3(µs)
Packet mirr.	75	16.9	5.6 (5.7)	75	16.3	14.4 (17)
Header mirr.	65	14.9	5.6 (5.7)	65	14.3	14.4 (17)
P4-metadata	110	0	5.6 (5.7)	110	0	14.4 (17)

6.3 Real-Time *DAD* with P4-Enabled Switches

We now assume to deploy the ML-based classifier to perform real-time *DAD* and evaluate the impact of performing features extraction at the data plane in the P4-enabled switches. To do so, we consider three different scenarios, where the P4 switch provides different types of traffic information data to the ML classifier, namely: (a) *packet mirroring*: the P4 switch forwards the received packets to the ML-based module, which performs features extraction before making predictions, (b) *header mirroring*: the P4 switch forwards only the headers of the received packets, and (c) *P4-metadata extraction*: the P4 switch elaborates the received packets and extracts

proper "metadata" (i.e., the features), which are sent to the ML classifier. Then, for each case, we evaluate three latency contributions, namely:

- t_1: time needed by the P4 switch for packet processing, i.e., to elaborate packets and send traffic information for a single window to the attack detection module;
- t_2: time needed for window features extraction, either performed in the P4 switch or in the ML-assisted *DAD* module;
- t_3: time needed by the ML classifier to perform window classification, based on the extracted features.

Note that, in the three scenarios described above, time contribution t_3 does not change but only depends on the adopted ML algorithm (i.e., it corresponds to the test time discussed in Sect. 6.1). On the contrary, contributions t_1 and t_2 may vary according to the amount of processing performed at the data plane by the P4 switches, which, instead of simple traffic mirroring, can also accomplish features extraction. A summary of the three scenarios and the notation for the three time contributions is summarized in Fig. 9.

To perform this analysis we consider RF and SVM algorithms as described above, since they provide the best results in terms of accuracy and test time. For all cases we consider windows of duration $T = 1$ s and datasets generated with $\delta = 0.01$ s, and show numerical results in Table 5. The latency values have been obtained by averaging over several runs and we report in Table 5 the average values. In particular, the latencies introduced by the P4 switch are evaluated using a BMV2 running on Linux Box, Intel Xeon CPU E5-2620 v2 @ 2.10GHz, RAM 16GB and 10Gigabit Ethernet optical interfaces, and are measured using the Spirent N4U Traffic generator and analyzer and injecting traffic profile as in Table 1. According to the considered value of T and the overall traffic profiles, the P4 switch is programmed with a traffic window of 10^5 packets. Moreover, latency contributions t_2 due for features extraction for the cases in Fig. 9a, b have been calculated by feeding a customized python-based script with *.pcap* traces and executing it on a desktop with 8×2 GHz processor and 8 GB of RAM.[6]

For both RF and SVM a significant time reduction is obtained in the *P4-metadata extraction* scenario, due to the time savings obtained by extracting windows features directly in the programmable switches. A P4 switch is able to extract features in around 110 µs (time contribution t_1), which is extremely low if compared to time contribution t_2 in *Packet mirroring* and *Header mirroring* scenarios, which ranges between 14.3 and 16.9 s. Moreover, the additional time required for feature extraction at the P4 switch (i.e., contribution t_2 in the *P4-metadata* case) is negligible if compared to both *Packet mirroring* and *Header mirroring*. Finally, as expected, the classification time contribution t_3 does not depend on the switch scenario, but only on the adopted ML algorithm or the *Standalone* and *Correlated DAD* architecture,

[6] Note that, although the features extraction can be optimized to reduce latency even further, further advantages in terms of port line rate and processing requirements might be obtained in the *P4-metadata extraction* scenario.

and equals 5.6 and 14.4 µs for RF and SVM in the *Standalone DAD*, whilst 5.7 and 17 µs for the same algorithms in the *Correlated DAD*.

7 Conclusion

In this paper we evaluated ML-assisted DDoS attack detection frameworks for application in SDN environment considering *Standalone* and *Correlated DAD* architectures. Leveraging the potential of data-plane programmability enabled by P4 language, we evaluated how detection latency is reduced when performing features extraction at P4 switches. To do so, we compared different ML classifiers in terms of accuracy and computational time, and deployed the algorithms in a real-time scenario in which the P4 switch provides different types of data to the ML classifiers, namely, *packet mirroring*, *header mirroring*, and *P4-metadata extraction*. Numerical results show that attack detection can be performed with classification accuracy, precision, recall and F1-score higher than 98% in most cases, and with drastic time reduction, down to less than 200 µs, in case P4 is used for features extraction. As a future work, we plan to investigate attack-type identification by developing multi-class ML classifiers, and implementing attack detection exploiting ML algorithms which leverage historical data, such as Recurrent Neural Networks.

Acknowledgements This work has received funding from the ECSEL Joint Undertaking (JU) BRAINE Project, under grant agreement No 876967. The JU receives support from the European Union's H2020 research and innovation programme and from Italy Ministry of Education, University and Research (MIUR).

Funding Open access funding provided by Politecnico di Milano within the CRUI-CARE Agreement.

References

1. Kreutz, D., Ramos, F.M.V., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S.: Software-defined networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015)
2. TCP Syn Flooding Attacks and Common Mitigations: IETF RFC **4987** (2007)
3. Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., Walker, D.: P4: Programming protocol-independent packet processors. ACM SIGCOMM Comput. Commun. Rev **44**(3), 87–95 (2014)
4. Musumeci, F., Ionata, V., Paolucci, F., Cugini, F., Tornatore, M.: Machine-learning-assisted DDoS attack detection with P4 language. In: IEEE International Conference on Communications (ICC) 2020, pp. 1–6. Dublin, Ireland (2020)

5. Zargar, S.T., Joshi, J., Tipper, D.: A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. IEEE Commun. Surv. Tutor. **15**(4), 2046–2069 (2013)
6. Hoyos Ll, M.S., Isaza E, G.A., Vélez, J.I., Castillo O, L.: Distributed denial of service (DDoS) attacks detection using machine learning prototype. In: Omatu, S., Semalat, A., Bocewicz, G., Sitek, P., Nielsen, I.E., García García, J.A., Bajo, J. (eds.) Distributed Computing and Artificial Intelligence, 13th International Conference, pp. 33–41. Springer, UK (2016)
7. Ramamoorthi, A., Subbulakshmi, T., Shalinie, S.M.: Real time detection and classification of DDoS attacks using enhanced SVM with string kernels. In: International Conference on Recent Trends in Information Technology (ICRTIT) 2011, pp. 91–96 (2011)
8. Sahoo, K.S., Tripathy, B.K., Naik, K., Ramasubbareddy, S., Balusamy, B., Khari, M., Burgos, D.: An evolutionary SVM model for DDOS attack detection in software defined networks. IEEE Access **8**, 132502–132513 (2020)
9. Saini, P.S., Behal, S., Bhatia, S.: Detection of DDoS attacks using machine learning algorithms. In: 2020 7th International Conference on Computing for Sustainable Global Development (INDIACom), pp. 16–21 (2020)
10. Subbulakshmi, T., Bala Krishnan, K., Shalinie, S.M., Anand Kumar, D., Ganapathi Subramanian, V., Kannathal, K.: Detection of DDoS attacks using Enhanced Support Vector Machines with real time generated dataset. In: Third International Conference on Advanced Computing 2011, pp. 17–22 (2011)
11. Ye, J., Cheng, X., Zhu, J., Feng, L., Song, L.: A DDoS attack detection method based on SVM in software defined network. Secur. Commun. Netw. **2018**, 1–8 (2018)
12. Bhardwaj, A., Mangat, V., Vig, R.: Hyperband tuned deep neural network with well posed stacked sparse autoencoder for detection of DDoS attacks in cloud. IEEE Access **8**, 181916–181929 (2020)
13. Sumathi, S., Karthikeyan, N.: Detection of distributed denial of service using deep learning neural network. Springer Journal of Ambient Intelligence and Humanized Computing (2020)
14. Yuan, X., Li, C., Li, X.: DeepDefense: identifying DDoS attack via deep learning. In: IEEE International Conference on Smart Computing (SMARTCOMP) 2017, pp. 1–8 (2017)
15. Aljuhani, A.: Machine learning approaches for combating distributed denial of service attacks in modern networking environments. IEEE Access **9**, 42236–42264 (2021)
16. AlMomin, H., Ibrahim, A.A.: Detection of distributed denial of service attacks through a combination of machine learning algorithms over software defined network environment. In: 2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), pp. 1–4 (2020)
17. Correa, J.H., Ciarelli, P.M., Ribeiro, M.R.N., Villaca, R.S.: ML-Based DDoS detection and identification using native cloud telemetry macroscopic monitoring. J. Netw. Syst. Manag. **2** (2021)
18. Gu, Y., Li, K., Guo, Z., Wang, Y.: Semi-supervised K-means DDoS detection method using hybrid feature selection algorithm. IEEE Access **7**, 64351–64365 (2019)
19. Mhamdi, L., McLernon, D., El-moussa, F., Raza Zaidi, S.A., Ghogho, M., Tang, T.: A deep learning approach combining autoencoder with one-class SVM for DDoS attack detection in SDNs. In: 2020 IEEE Eighth International Conference on Communications and Networking (ComNet), pp. 1–6 (2020)
20. de Miranda Rios, V., Inacio, P.R., Magoni, D., Freire, M.M.: Detection of reduction-of-quality DDoS attacks using Fuzzy Logic and machine learning algorithms. Comput. Netw. **186**, 107792 (2021)
21. Simpson, K.A., Rogers, S., Pezaros, D.P.: Per-host DDoS mitigation by direct-control reinforcement learning. IEEE Trans. Netw. Serv. Manage. **17**(1), 103–117 (2020)
22. Abou El Houda, Z., Khoukhi, L., Senhaji Hafid, A.: Bringing intelligence to software defined networks: mitigating DDoS attacks. IEEE Trans. Netw. Serv. Manag. **17**(4), 2523–2535 (2020)
23. Alamri, H.A., Thayananthan, V.: Bandwidth control mechanism and extreme gradient boosting algorithm for protecting software-defined networks against DDoS attacks. IEEE Access **8**, 194269–194288 (2020)
24. Dinh, P.T., Park, M.: BDF-SDN: A big data framework for DDoS attack detection in large-scale SDN-based cloud. In: 2021 IEEE Conference on Dependable and Secure Computing (DSC), pp. 1–8 (2021)
25. Dong, S., Sarem, M.: DDoS attack detection method based on improved KNN with the degree of DDoS attack in software-defined networks. IEEE Access **8**, 5039–5048 (2020)

26. Kokila, R.T., Thamarai Selvi, S., Govindarajan, K.: DDoS detection and analysis in SDN-based environment using support vector machine classifier. In: International Conference on Advanced Computing (ICoAC) 2014, pp. 205–210 (2014)
27. Mousavi, S.M., St-Hilaire, M.: Early detection of DDoS attacks against software defined network controllers. J. Netw. Syst. Manag. **26**, 573–591 (2018)
28. Perez-Díaz, J.A., Valdovinos, I.A., Choo, K.K.R., Zhu, D.: A flexible SDN-based architecture for identifying and mitigating low-rate DDoS attacks using machine learning. IEEE Access **8**, 155859–155872 (2020)
29. Phan, T.V., Nguyen, T.G., Dao, N.N., Huong, T.T., Thanh, N.H., Bauschert, T.: DeepGuard: efficient anomaly detection in SDN with fine-grained traffic flow monitoring. IEEE Trans. Netw. Serv. Manag. **17**(3), 1349–1362 (2020)
30. Ravi, N., Shalinie, S.M.: Learning-driven detection and mitigation of DDoS attack in IoT via SDN-cloud architecture. IEEE Internet Things J. **7**(4), 3559–3570 (2020)
31. Sudar, K.M., Beulah, M., Deepalakshmi, P., Nagaraj, P., Chinnasamy, P.: Detection of Distributed Denial of Service Attacks in SDN using Machine learning techniques. In: 2021 International Conference on Computer Communication and Informatics (ICCCI), pp. 1–5 (2021)
32. Tan, L., Pan, Y., Wu, J., Zhou, J., Jiang, H., Deng, Y.: A new framework for DDoS attack detection and defense in SDN environment. IEEE Access **8**, 161908–161919 (2020)
33. Zhijun, W., Qing, X., Jingjie, W., Meng, Y., Liang, L.: Low-rate DDoS attack detection based on factorization machine in software defined network. IEEE Access **8**, 17404–17418 (2020)
34. Zhu, L., Tang, X., Shen, M., Du, X., Guizani, M.: Privacy-preserving DDoS attack detection using cross-domain traffic in software defined networks. IEEE J. Sel. Areas Commun. **36**(3), 628–643 (2018)
35. Afek, Y., Bremler-Barr, A., Shafir, L.: Network anti-spoofing with SDN data plane. In: IEEE Conference on Computer Communications (INFOCOM) 2017, pp. 1–9 (2017)
36. Sviridov, G., Bonola, M., Tulumello, A., Giaccone, P., Bianco, A., Bianchi, G.: LODGE: LOcal Decisions on Global statEs in prograna naable data planes. In: IEEE Conference on Network Softwarization and Workshops (NetSoft) 2018, pp. 257–261 (2018)
37. Lapolli, A.C., Adilson Marques, J., Gaspary, L.P.: Offloading real-time DDoS attack detection to programmable data planes. In: IFIP/IEEE Symposium on Integrated Network and Service Management (IM) 2019, pp. 19–27 (2019)
38. Ilha, A.d.S., Lapolli, A.C., Marques, J.A., Gaspary, L.P.: Euclid: A fully in-network, P4-based approach for real-time DDoS attack detection and mitigation. IEEE Transactions on Network and Service Management, pp. 1–1 (2020)
39. Paolucci, F., Civerchia, F., Sgambelluri, A., Giorgetti, A., Cugini, F., Castoldi, P.: P4 edge node enabling stateful traffic engineering and cyber security. IEEE/OSA J. Opt. Commun. Networking **11**(1), A84–A95 (2019)
40. Febro, A., Xiao, H., Spring, J.: Distributed SIP DDoS defense with P4. In: IEEE Wireless Communications and Networking Conference (WCNC) 2019, pp. 1–8 (2019)
41. Zhang, M., Li, G., Wang, S., Liu, C., Chen, A., Hu, H., Gu, G., Li, Q., Xu, M., Wu, J.: Poseidon: mitigating volumetric ddos attacks with programmable switches. In: Proceedings of the Network and Distributed System Security Symposium (NDSS) 2020, pp. 1–18 (2020)
42. Friday, K., Kfoury, E., Bou-Harb, E., Crichigno, J.: Towards a unified in-network DDoS detection and mitigation strategy. In: IEEE Conference on Network Softwarization (NetSoft) 2020, pp. 218–226 (2020)
43. Jalili, R., Imani-Mehr, F., Amini, M., Shahriari, H.R.: Detection of Distributed Denial of Service Attacks Using Statistical Pre-processor and Unsupervised Neural Networks. Springer, Berlin (2005)
44. Braga, R., Mota, E., Passito, A.: Lightweight DDoS flooding attack detection using NOX/OpenFlow. In: IEEE Local Computer Network Conference 2010, pp. 408–415 (2010)
45. Kalkan, K., Altay, L., Gür, G., Alagöz, F.: JESS: joint entropy-based DDoS defense scheme in SDN. IEEE J. Sel. Areas Commun. **36**(10), 2358–2372 (2018)
46. Cugini, F., Gunning, P., Paolucci, F., Castoldi, P., Lord, A.: P4 in-band telemetry (INT) for latency-aware VNF in metro networks. In: Optical Fiber Communication Conference (OFC) 2019, p. M3Z.6. Optical Society of America (2019)
47. Spirent Test Center. https://www.spirent.com/products/testcenter (2020). Accessed Oct 2020
48. MAWI Working Group Traffic Archive: Packet traces from WIDE backbone. https://mawi.wide.ad.jp/mawi/ (2015)

49. Rüth, J., Poese, I., Dietzel, C., Hohlfeld, O.: A first look at quic in the wild. In: Beverly, R., Smaragdakis, G., Feldmann, A. (eds.) Passive and Active Measurement, pp. 255–268. Springer, Cham (2018)
50. Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., Yang, F., Kouranov, F., Swett, I., Iyengar, J., Bailey, J., Dorfman, J., Roskind, J., Kulik, J., Westin, P., Tenneti, R., Shade, R., Hamilton, R., Vasiliev, V., Chang, W.T., Shi, Z.: The quic transport protocol: design and internet-scale deployment. In: Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 17, p. 183–196. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3098822.3098842
51. Genome, I.M.I.X.: Specification of variable packet sizes for additional testing. IETF RFC **6985** (2013)
52. Benchmarking Methodology for Network Interconnect Devices: IETF RFC **2544** (1999)
53. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in statistics New York (2008)

Francesco Musumeci received the Ph.D. degree in Information Engineering from Politecnico di Milano, Italy, in 2013, where he is currently an Assistant Professor with the Department of Electronics, Information and Bioengineering since 2016. His current research interests include Machine-Learning-assisted networking, design and optimization of optical networks, and network disaster resilience. He is author of more than 90 papers in international journals and conference proceedings, 2 book chapters and 1 patent in the area of communication networks, published, and is co-winner of three best paper awards from IEEE sponsored conferences. He uses to serve as a TPC member and/or reviewer for several IEEE/OSA conferences as well as IEEE/OSA, Springer and Elsevier journals since 2010.

Ali Can Fidanci received the B.Sc. degree in Electronics and Communication Engineering from Istanbul Technical University, Turkey, in 2017 and the M.Sc. degree in Telecommunication Engineering from Politecnico di Milano, Italy in 2020, where he focused his research interests on Machine Learning for cybersecurity. Currently, he works as system engineer at ASELSAN, Turkey.

Francesco Paolucci received the Laurea degree in telecommunications engineering from the University of Pisa in 2002, and the Ph.D. degree from Scuola Superiore Sant'Anna, Pisa, in 2009. In 2008 he was granted a research Merit Scholarship at the Istitut National de le Recherche Scientifique (INRS), Montreal, Quebec, Canada. Currently, he is Senior Researcher at CNIT, Pisa Italy. His main research interests are in the field of network control plane, orchestration for edge/cloud platforms, traffic engineering, network disaggregation, advanced network telemetry, SDN/P4 data plane programmability. He has been involved in many European research projects on next generation control networking (E-Photon/ONe+, BONE, NOBEL, STRONGEST, IDEALIST, PACE, 5GEx, 5GTRANSFORMER, METROHAUL, 5Growth, BRAINE). He is co-author of 2 IETF Internet Drafts, more than 170 publications in international journals, conference proceedings and book chapters, and filed 4 international patents. He is Associate Editor of the IEEE/OSA Journal of Optical Communications and Networking (JOCN).

Filippo Cugini is Head of Research Area at CNIT, Pisa, Italy. His main research interests include theoretical and experimental studies in the field of packet and optical networking. He is co-author of 14 patents and more than 250 international publications.

Massimo Tornatore is currently an Associate Professor with the Department of Electronics, Information, and Bioengineering, Politecnico di Milano. He also holds an appointment as Adjunct Professor at University of California, Davis, USA and as visiting professor at University of Waterloo, Canada. His research interests include performance evaluation, optimization and design of communication networks (with an emphasis on the application of optical networking technologies), cloud computing, and machine learning application for network management. In these areas, he co-authored more than 400 peer-reviewed conference and journal papers (with 19 best paper awards), 2 books and 1 patent. He is a member of

the Editorial Board of IEEE Communication Surveys and Tutorials, IEEE Communication Letters, IEEE Transactions on Network and Service Management, and Elsevier Optical Switching and Networking. He is active member of the technical program committee of various networking conferences such as INFOCOM, OFC, ICC, and GLOBECOM. He acted as technical program chair of ONDM 2016 and DRCN 2017 and DRCN 2019 conferences. He has participated in several EU R&D projects (among others FP7 COMBO, H2020 MEtroHaul, and Cost Action RECODIS) as well as in several projects in USA, Canada and Italy.

Authors and Affiliations

Francesco Musumeci[1] · Ali Can Fidanci[1] · Francesco Paolucci[2,3] · Filippo Cugini[3] · Massimo Tornatore[1]

✉ Francesco Musumeci
francesco.musumeci@polimi.it

[1] Politecnico di Milano, Milan, Italy

[2] Scuola Superiore Sant'Anna, Pisa, Italy

[3] CNIT, Pisa, Italy