Machine-learning-assisted DDoS attack detection with P4 language

Francesco Musumeci*†, Valentina Ionata*, Francesco Paolucci‡, Filippo Cugini§, Massimo Tornatore*
*Politecnico di Milano, Milan, Italy. ‡Scuola Superiore Sant'Anna, Pisa, Italy.
§Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Pisa, Italy
†Corresponding author: francesco.musumeci@polimi.it

***Abstract*—While Software Defined Networking (SDN) provides well-known advantages in terms of network automation, flexibility and resources utilization, it has been observed that SDN controllers may represent critical points of failure for the entire network infrastructure, especially when they are targeted by malicious cyber attacks such as Distributed Denial of Service (DDoS). To address this issue, in this paper we exploit stateful data planes, as enabled by P4 programming language, where switches maintain persistent memory of handled packets to perform attack detection directly at the data plane, with only marginal involvement of the SDN controllers. As machine learning (ML) is recognized as primary anomaly detection methodology, we perform DDoS attack detection using a ML-based classification and compare different ML algorithms in terms of classification accuracy and train/test duration. Moreover, we combine ML and P4-enabled stateful data planes to design a real-time DDoS attack detection module, which we evaluate in terms of latency required for the detection. Three real-time scenarios are considered, where P4-enabled switches elaborate the received packets in different ways, namely, *packet mirroring*, *header mirroring*, and *P4-metadata extraction*. Numerical results show significant latency reduction when P4 is adopted.**

I. Introduction

Software Defined Networking (SDN) has attracted the attention of many researchers and practitioners in the last decade [1] as it provides several advantages in comparison to traditional legacy networking, e.g., automated network control, more efficient resources utilization, higher flexibility and faster reconfigurability. Such advantages are mainly brought by the logically-centralized network view, enabled by a dedicated control plane, which consists of several SDN controllers and is decoupled from the data plane. However, due to their central role, the SDN controllers represent a vulnerable target for malicious cyber attacks such as Distributed Denial of Service (DDoS), where non-legitimate data packets are sent overload the controllers, thus inhibiting proper functioning of data plane devices. Moreover, even when the DDoS attack victim is not the SDN controller, it might be overflooded due to the default forwarding policies implemented in SDN switches. In view of this, we consider that the traditional data plane programmability enabled by SDN can be augmented with *stateful data planes* to perform DDoS attack detection (*DAD*). In such scenario, switches maintain persistent states (i.e., the have *memory* of handled packets) so that complex in-network per-packet processing can be performed with no need for cooperating with the control plane to make forwarding decisions. In this study, we observe that stateful data planes, besides reducing the data forwarding latency, can also be used to mitigate DDoS attacks by limiting the amount of malicious packets forwarded to the SDN controller, thus preventing its overflooding. Stateful data planes can be implemented, e.g., by P4 [2], which is the data plane programming language considered in this paper.

A. Related work

DAD has been widely investigated in literature [3]. It typically relies on machine learning (ML), due to its ability to automatically learn hidden patterns in traffic flows and to detect anomalies. For example, Ref. [4] proposes a Support Vector Machine (SVM) classifier to perform *DAD* in SDN networks, whereas in [5] SVM classification is improved by using a string kernel to perform attack detection in real time. SVM classifiers are used also in [6], where several types of DDoS attacks are considered. Authors in [7] have used deep recurrent neural networks to learn patterns from network traffic and perform *DAD*. Ref. [8] has considered an improved version of k-nearest-neighbour (KNN) to contrast DDoS attacks in SDN networks, while preserving operators privacy across different network domains.

Leveraging stateful data planes for *DAD* is a very recent research trend. In [9] a model for the coordination of stateful switches for *DAD* is presented, whereas Ref. [10] proposes a distributed architecture of stateful switches to mitigate the attacks as an alternative to centralized solutions (as it might occur in SDN) which suffer from possible bottlenecks in computational resources. In [11] authors leverage P4 to perform traffic inspection for real-time *DAD*, similarly to [12], where the authors focused on a simple strategy to contrast Transmission Control Protocol (TCP) flood port scan attacks.

Following the idea in [12], in this paper we specifically focus on TCP flood attacks and combine ML and P4 to effectively perform *DAD*. More precisely, we combine the ML capability to elaborate data with the potential of stateful data planes in collecting traffic information, to reduce the risk of SDN controller overflooding. To the best of our knowledge, this is the first time that these two aspects are combined in the context of cybersecurity.

B. Paper contribution and organization

In this paper, 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 we emulate a situation where the *DAD* is performed in real time in a ML-based module which directly interacts with the P4-enabled switch. Specifically, the main contributions of this paper are as follows:

- we compare different ML algorithms for *DAD*, in terms of accuracy and algorithm complexity, i.e., duration of algorithm training and prediction time;
- we provide P4 code for the extraction of features to be used in the ML classifiers;
- based on the ML algorithms analysis and leveraging P4 language, we evaluate 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 paper is organized as follows. In Sec. II we provide background on DDoS attack and P4 language. In Sec. III we overview the ML-assisted *DAD* framework and detail the characteristics of the ML algorithms used to perform the attack detection. In Sec. IV we describe the P4 code adopted to extract metadata from packet flows at the P4-enabled switch and provide concluding remarks in Sec. VI.

II. Background

A. DDoS attacks

The focus of this paper is on the ML-assisted detection of DDoS attacks. In general, Denial of Service (DoS) attacks aim at inhibiting a server, host or application by repeatedly sending malicious traffic to the victim, to overload its computing and/or network resources. In their distributed form, DDoS attacks are generated by multiple sources with different IP addresses, so that using an IP-addresses blacklist at the victim is not adequate to block the attacks. The most common DDoS attacks are usually grouped in the following categories: TCP flood, UDP flood, ICMP flood and HTTP flood, according to the protocol used to perform the attack. In particular, in this paper we focus on TCP flood attacks, which exploit TCP connection initiation packets to target the victim. Usually the attacker sends multiple SYN requests from multiple spoofed IP addresses, but it does not respond to the victim SYN-ACK packet. This way, the victim's system is halted while waiting for the ACK messages from all the senders, which would terminate the TCP connections setup.

The defense mechanisms against network/transport-level DDoS attacks are usually classified into three main categories, based on the location of the detection engine [3], i.e., *source-based*, *destination-based* and *network-based* mechanisms. As our objective is to prevent DDoS attacks directly at the SDN network switches, in this paper we focus on network-based detection mechanisms considering TCP flood attacks only.

B. P4 language

Due to the centralized nature of SDN, cyber attacks may seriously affect the functioning of SDN controllers and hence of the entire network. Therefore, it is important to limit the direct involvement of SDN controllers in *DAD*. In fact, a proper attack detection requires time-consuming and expensive deep packet inspection, which is not desirable to be implemented in SDN controllers. To offload SDN controllers from this heavy computation, in this paper we leverage data plane programmability by adopting the P4 (Programming Protocol-independent Packet Processors) open source language [2]. P4 language is a high-level programming language for routers and switches, designed to allow programming of packet forwarding planes. In other words, P4 enables packet processing at forwarding elements such as hardware or software switches, network interface cards, routers, etc., and it is not constrained by the traditional fixed-functions protocol stack. A P4 program is composed by the following main components:

- *Parsers* identify the stack of allowed protocols and fields adopted in the program. Typically, they contain the names of the used headers and their size in bits; optional field annotations allow constraints on value ranges or maximum lengths for variable-sized fields.
- *Tables* include the packet processing rules in the form of "match-action". When packets are processed by the P4 program, the ingress pipeline is executed to look for the matching rule(s) which fits the incoming packet. Several types of rules can be defined, including field updates, egress port selection, etc.
- *Control Programs (ingress/egress)* determine the order of match-action tables that are applied to a packet. A simple program describes the flow of control between match-action tables.

In addition, P4 defines programmable packet metadata, used to associate extra information to the packet itself (e.g., timestamps, features, state), and stateful objects (e.g., meters, counters, registers) that may be used to perform context-based processing and implement Finite State Machines.

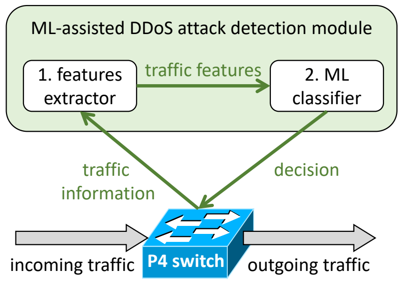
III. ML-assisted DDoS attack detection

A. Framework overview

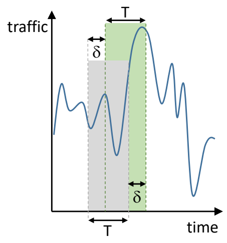
As depicted in Fig. 1(a), the DDoS attack detection module periodically receives traffic information from the P4 switch and, based on such information, performs attack detection so that the switch can decide if packets can be forwarded or not. To this end, we model the DDoS attack detection into a ML classification problem. Given the traffic (i.e., one or more packets arriving at the P4 switch) observed in a certain time frame with pre-defined duration, the detection module outputs a decision for the observed traffic, i.e., a label "attack" or "no-attack", to indicate that in the considered time frame an attack is present or not, respectively[1].

The detection module is generally constituted by two operational blocks, i.e., *1) features extractor* and *2) ML classifier*. However, according to the operations performed at the P4 switch, the detection module can be highly simplified or even

[1]Note that, more advanced detection can be performed to distinguish different types of DDoS attacks, in which case a multi-class classifier is developed. We plan to address this analysis in a future work.



(a) Detection framework and functional blocks



(b) Traffic window

Fig. 1. ML-assisted DDoS attack detection characteristics.

integrated directly at the data plane in the P4 switch. To this end, in this paper we also perform an analysis to evaluate the additional latency introduced by the attack detection module comparing different situations in which various types of traffic information are exchanged between the detection module and the P4 switch. To perform attack detection we consider traffic windows of duration T (as qualitatively shown in Fig. 1(b)). We assume that traffic information is periodically sent from the P4 switch to the detection module on a sliding-window basis (e.g., every δ seconds), so that each window contains information of the last T seconds and is classified independently from the other windows. The various classification outputs can be then exploited to perform operator-specific actions, e.g., drop the packets after a number of subsequent windows are classified as containing an attack.

B. ML classifiers

To implement the classifiers for *DAD* we consider three different ML algorithms, namely, Random Forest (RF), K-Nearest Neighbours (KNN) and Support Vector Machine (SVM). The comparison between the various algorithms has been carried out to devise the most appropriate solution in terms of classification accuracy and algorithm complexity, i.e., training duration and prediction time[2].

The classifiers in this paper have been developed in the form of binary classifiers, i.e., each time a window is classified, one of the following two labels is assigned to the window: *"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 details of the features extracted from the time window.

For each traffic window of duration T starting at a generic time instant t, we consider the following features:

- *Average length, Len(t)*: the average size in bytes of packets in time window $(t,\, t+T)$;
- *TCP ratio,* $R_{TCP}(t)$: the percentage of TCP packets out of the total in time window $(t,\, t+T)$;
- *UDP ratio,* $R_{UDP}(t)$: is similar to R_{TCP} and represents the percentage of UDP packets;
- *TCP-UDP ratio,* $R_{TU}(t)$: is the ratio between TCP and UDP packets in time window $(t, t{+}T)$. If no UDP packet

[2]Note that, in the ML environment, time-series data as that considered in this paper is often handled by means of artificial neural networks, especially in the form of recurrent neural networks. In this paper we do not consider such type of ML classifiers, which we will investigate in future work.

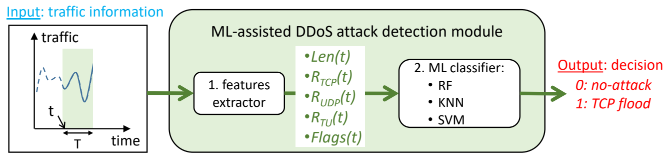


Fig. 2. Window features extraction and classification.

is present in the window, we set this feature equal to a large finite number;

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

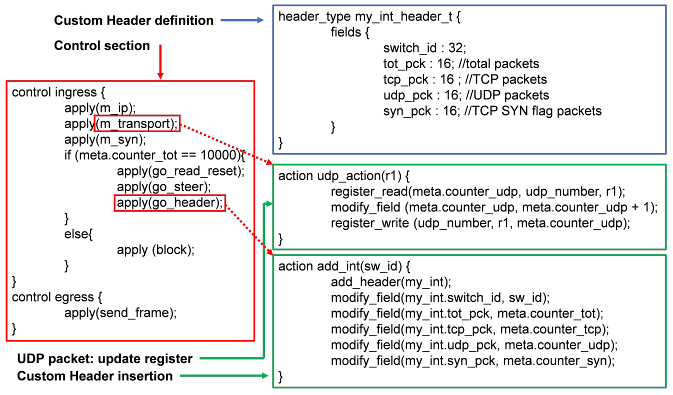
Although several traffic features have been adopted in literature to perform *DAD* [13], as we focus on TCP flood attacks, the previous features have been considered as the most appropriate for our analysis.

IV. Using P4 language for attack detection in data plane

The efficiency of ML classifiers to perform real-time *DAD* can be further improved by elaborating traffic features directly at the data plane, in order to offload the features extraction process from the *DAD* module. To do so, we exploit the potential of stateful data planes enabled by P4 language to deploy P4 switches implementing packet mirroring, header-mirroring and metadata extraction. In this section we describe the P4 switch performing traffic stateful analysis and reporting statistics to the ML-assisted *DAD* module under the form of in-band telemetry function (i.e., P4 metadata extraction version). Two specific P4 features have been exploited to this end:

- *Stateful objects processing*, in particular programmable registers storing and updating the number of selected packets occurrences within a traffic window;
- *Extra header definition*, used to provide the analysis results to the detection module.

Figure 3 shows an excerpt of selected key parts of the P4 code used to deploy the switch. The P4 code defines four registers, used to store the occurrences of IP, UDP, TCP and SYN packets. Moreover, besides standard L3 and L4 headers, it defines a 12-byte long custom header (i.e., `my_int_header_t`) including a `switch_id` and four 2-byte long fields reporting the number of total, UDP, TCP, TCP with SYN flag packets, respectively. The P4 switch program performs the following procedure. First, P4 extracts the standard headers of each traffic packet. In the control ingress pipeline, it updates the value of the internal counters based on their existence and the header values, resorting to three flow tables (i.e., `m_ip` for IP, `m_transport` for UDP/TCP and `m_syn` for SYN flag detection). Then, the same ingress pipeline is subject to a conditional constraint. In the case of traffic window end (i.e., reached number of maximum allowed packets in the window, stored in the packet metadata `meta.counter_tot`), the custom extra-header is inserted into the packet (table `go_header`, the register values are copied into the respective header fields and the resulting packet is sent out towards the detection module, triggering a reset of the internal registers (table `go_read_reset`). If the window



```
header_type my_int_header_t {
        fields {
                switch_id : 32;
                tot_pck : 16; //total packets
                tcp_pck : 16 ; //TCP packets
                udp_pck : 16; //UDP packets
                syn_pck : 16; //TCP SYN flag packets
        }
}
```

```
control ingress {
        apply(m_ip);
        apply(m_transport);
        apply(m_syn);
        if (meta.counter_tot == 10000){
                apply(go_read_reset);
                apply(go_steer);
                apply(go_header);
        }
        else{
                apply (block);
        }
}
control egress {
        apply(send_frame);
}
```

```
action udp_action(r1) {
        register_read(meta.counter_udp, udp_number, r1);
        modify_field (meta.counter_udp, meta.counter_udp + 1);
        register_write (udp_number, r1, meta.counter_udp);
}
```

```
action add_int(sw_id) {
        add_header(my_int);
        modify_field(my_int.switch_id, sw_id);
        modify_field(my_int.tot_pck, meta.counter_tot);
        modify_field(my_int.tcp_pck, meta.counter_tcp);
        modify_field(my_int.udp_pck, meta.counter_udp);
        modify_field(my_int.syn_pck, meta.counter_syn);
}
```

Fig. 3. P4 code excerpts: control section, custom header and register update/header insertion actions.

is not terminated, packets follow another sequence of flow tables. This way, only the last packet in each window includes the whole window statistics inside the custom header and provides them to the detection module. Fig. 3 also shows the definition of two P4 actions. The first operates on UDP packets inside `m_transport` table, in order to update register value. The register is first read, then modified and written utilizing a packet metadata. The second action operates inside the `go_header` table and inserts a new custom header on the packet, copying the packet statistics stored in the metadata inside each header field and the switch id.

V. Numerical Results

A. Case study and settings

ML algorithms have been implemented with Python-based scripts using *keras* and *sklearn* libraries on a desktop with 8$\times$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 [14]. We generate realistic traffic traces for 30 minutes, where a TCP flood attack flow at a constant bit rate of 10, 50, or 100 kbit/s is added to regular background traffic at 10 Mbit/s. Average duration of the attacks is 10 seconds, whereas background traffic is composed by three different flows, i.e., 4.5 Mbit/s TCP traffic, 3.8 Mbit/s UDP traffic and 1.7 Mbit/s IP traffic, with packet length following iMix distribution [14]. To label our dataset, we consider all the windows of duration T and assign label *"1: TCP flood"* only if it contains at least one packet belonging to the TCP flood attack, otherwise the window is assigned label *"0: no-attack"*. Note that, since in our experiment we tune the distance between two consecutive windows, i.e., parameter δ in Fig. 1(b), the total number of windows in the dataset depends on the value of δ, i.e., it ranges between 1800 and 36000 for the cases of $\delta = 1$s and $\delta = 0.05$s, respectively. Moreover, in the following we also do a sensitivity analysis on the window duration T, so for each pair (T, δ) a new dataset is generated for training and testing of our algorithm (see Tab. I).

B. ML algorithms performance evaluation

To identify the most appropriate solution for the *DAD* we perform a comparison between the various algorithms in terms of classification accuracy and algorithm complexity, i.e., training duration and prediction time. We also perform an analysis varying the window duration and shift (i.e., parameters T and δ in Tab. I) to investigate a balance between accuracy and complexity in the various cases. For all the ML algorithms we adopt k-fold cross-validation (with $k = 5$ in our case) to calculate average classification accuracy and to perform hyperparameters selection (e.g., to select the kernel in the SVM, the number of trees in RF and the number K of neighbors in KNN).

TABLE I
Traffic Parameters in the Dataset

Parameter	Value
Traces duration	30 m
TCP traffic bit rate	4.5 Mbit/s
UDP traffic bit rate	3.8 Mbit/s
IP traffic bit rate	1.7 Mbit/s
Attack traffic bit rate	100 kbit/s
Attack Type	SYN flood
Window duration	$T \in \{0.5; 1; 2; 5; 10\}$ seconds
Windows distance	$\delta \in \{0.05; 0.1; 0.2; 0.5; 1\}$ seconds

1) Impact of windows distance, δ: We start the analysis comparing the three algorithms for a fixed value of the window duration, i.e., $T = 1$s, and increasing distance between two consecutive windows (i.e., parameter δ). Results are shown in Fig. 4 in terms of (a) classification accuracy, (b) cross-validation time (training & test) and (c) test time, respectively.

As shown in Fig. 4(a), RF classifier outperforms SVM and KNN for all the values of δ, reaching a classification accuracy higher than 98.5% for $\delta = 0.05$s, which is 0.1% and 0.4% higher than classification accuracy of SVM and KNN, respectively, for the same value of δ. For all classifiers, classification accuracy decreases for increasing values of δ, due to the fact that a smaller dataset is used to train the classifiers. The accuracy decrease is more evident for the KNN algorithm, which provides accuracy lower than 97.5% for $\delta = 1$s. In Fig. 4(b) we show the cross-validation time for the three algorithms in the case of $T = 1$s and for increasing δ. This time is calculated as the total time required to perform the k-fold cross-validation (i.e., training and testing for $k = 5$ times in our case) of the algorithms for the various values of δ.Note that, for KNN algorithm, there is no real training phase as the classification is performed by calculating the weighted euclidean distance from the K nearest points. Therefore, the cross-validation time is mostly driven by the time needed to perform the classification (i.e., the test time). As expected, cross-validation time decreases for increasing δ, due to the smaller amount of elements (windows) in the dataset. This decrement is much more evident for SVM rather than the other algorithms, especially RF, which provides minimal decrease with increasing δ, although the cross-validation time is low (around 200 s) in all cases. Figure 4(c) shows the test time, i.e., the time required to perform the classification of one test set (in one of the $k = 5$ folds of the cross-validation phase). A similar trend of the one in Fig. 4(b) is observed also for the test time. More specifically, the value of δ does not affect

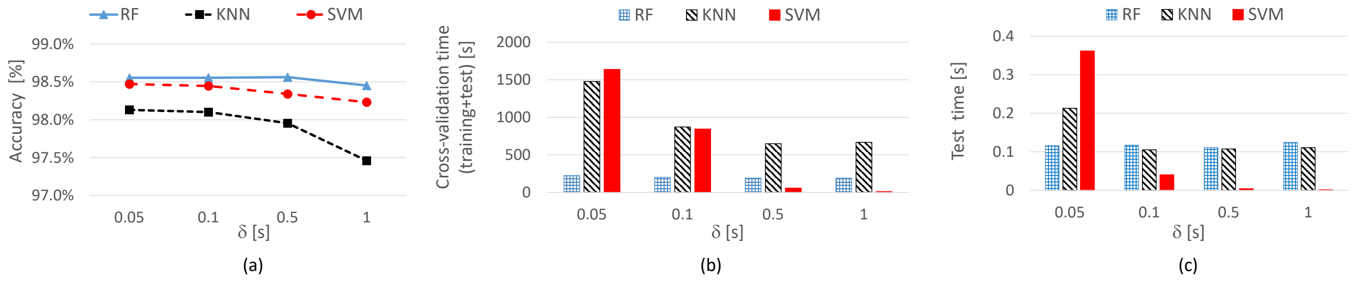


Fig. 4. Comparison between the different ML algorithms for fixed window duration ($T = 1$s) and varying δ: (a) Classification accuracy; (b) Cross-validation time (training & test); (c) Test time.

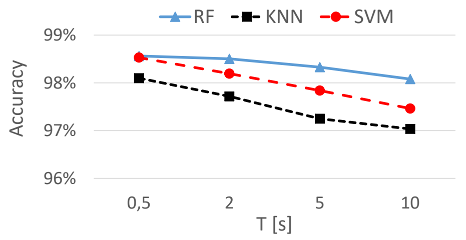


Fig. 5. Classification accuracy for the different ML algorithms for $\delta = 0.2$s and varying window duration T.

relevantly the test time for RF and KNN (in the last case, except for very low values of δ), whereas it has a high impact in decreasing complexity for SVM. This confirms that SVM and RF represent good candidates for the online *DAD*, as they provide a good trade-off between classification accuracy and algorithm complexity.

2) Impact of window duration, T: Figure 5 shows the classification accuracy for the three algorithms considering $\delta = 0.2$s and increasing values of window duration T. In all cases accuracy decreases for increasing T, mainly due to the fact that, in this case, the window duration is overdimensioned with respect to the attack, so that a lower quantity of attack packets is present in the window compared to the regular background traffic, and this makes it more difficult to identify the attacks signature. The accuracy of RF algorithm is the highest for all the values of T, up to 0.5% and 1% higher than SVM and KNN, respectively, and the accuracy decrease observed for increasing T is slower for the RF case in comparison to the other two algorithms, confirming the robustness of this algorithm to variations of the window parameters.

C. Real-time DAD in P4-enabled switch

After we evaluate the various ML algorithms, we assume to deploy the most suitable ones in real-time *DAD* framework and evaluate the impact of performing features extraction at the data plane. To do so, we evaluate the additional latency introduced at the data plane by the interaction between the P4 switch and the ML-based *DAD* module. In particular, 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. In this context we identify the following three latency contributions:

1) 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;
2) t_2: time needed for window features extraction, either performed in the P4 switch or in the ML-assisted *DAD* module;
3) t_3: time needed by the ML classifier to perform window classification, based on the extracted features.

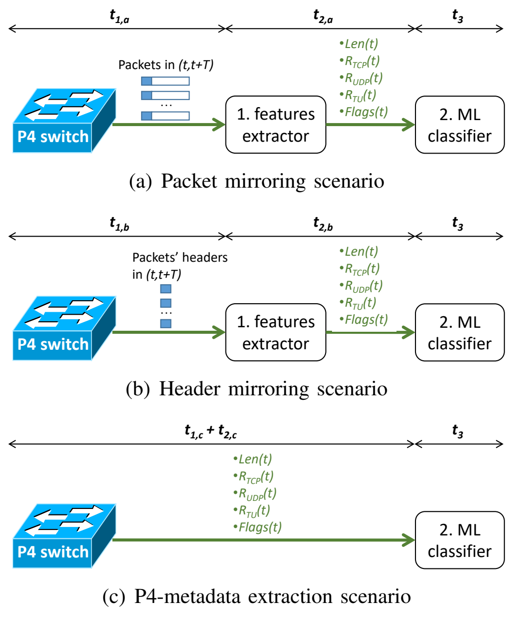


Fig. 6. Different scenarios and corresponding time contributions.

Note that, in the three scenarios *a), b)* and *c)* described above, time contribution t_3 does not change, whereas contributions t_1 and t_2 may vary due to the fact that the P4 switch performs more advanced packet processing compared to bare traffic mirroring and so the operation of features extraction can be simplified. A summary of the three scenarios and the notation for the three time contributions is summarized in Fig. 6.

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. We perform model selection for both algorithms to identify the hyperparameters providing the highest accuracy. Specifically, the RF adopts 10 different trees and a splitting criterion based on Gini impurity [15], whereas in the SVM case we adopt a Radial basis function kernel [15]. For both algorithms we consider as window

TABLE II
TIME CONTRIBUTIONS FOR DIFFERENT SCENARIOS

	RF			SVM		
	t_1	t_2	t_3	t_1	t_2	t_3
Packet mirr.	75 μs	16.9 s	0.1 s	75 μs	16.3 s	0.4 ms
Header mirr.	65 μs	14.9 s	0.1 s	65 μs	14.3 s	0.4 ms
P4-metadata	**110 μs**	**0 s**	**0.1 s**	**110 μs**	**0 s**	**0.4 ms**

parameters the values $T = 1$s and $\delta = 0.2$s, and obtain classification accuracy and AUC (Area Under the Receiver Operating Characteristic) always above 98%.

Numerical results are shown in Tab. II for the various cases. The latency values have been obtained by averaging over several runs and we report in Tab. II 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 Tab. I. On the other hand, the features extraction latency contribution (t_2) for the cases in Figs. 6(a) and 6(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[3]. As expected, for both ML algorithms a significant time reduction is obtained when the features extraction is performed at the data plane by the P4 switch (i.e., in the *P4-metadata extraction* scenario). Indeed, the P4 switch is able to extract features in around 110 μs (time contribution t_1), which is extremely low if compared to 14-16 seconds required by the external features extraction module (time contribution t_2) in both Packet mirroring and Header mirroring scenarios. It is worth noting that the additional time required to perform features extraction at the P4 switch is negligible if compared to both Packet mirroring and Header mirroring scenarios. Finally, as expected, the time contribution for the classification (time contribution t_3) does not depend on the switch scenario, but only on the adopted ML algorithm.

VI. CONCLUSION

We developed a ML-assisted DDoS attack detection framework for SDN networks, exploiting the potential of P4-enabled data plane programmability to perform features extraction at the data plane to drastically reduce the detection latency. To this end, 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 accuracy can be performed with accuracy higher than 98% in most cases and, in case P4 is used for features extraction, in less than 1 ms.

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

As a future work, we plan to investigate the possibility of implementing the entire attack detection process at the data plane, exploiting ML algorithms which leverage historical data, such as Recurrent Neural Networks.

ACKNOWLEDGEMENT

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).

REFERENCES

[1] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[2] 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," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, July 2014.

[3] S. T. Zargar, J. Joshi, and D. Tipper, "A Survey of Defense Mechanisms Against Distributed Denial of Service (DDoS) Flooding Attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 4Q 2013.

[4] Kokila RT, S. Thamarai Selvi, and K. Govindarajan, "DDoS detection and analysis in SDN-based environment using support vector machine classifier," in *International Conference on Advanced Computing (ICoAC) 2014*, Dec. 2014, pp. 205–210.

[5] A. Ramamoorthi, T. Subbulakshmi, and S. M. Shalinie, "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*, June 2011, pp. 91–96.

[6] T. Subbulakshmi, K. Bala Krishnan, S. M. Shalinie, D. Anand Kumar, V. Ganapathi Subramanian, and K. Kannathal, "Detection of DDoS attacks using Enhanced Support Vector Machines with real time generated dataset," in *Third International Conference on Advanced Computing 2011*, Dec. 2011, pp. 17–22.

[7] X. Yuan, C. Li, and X. Li, "DeepDefense: Identifying DDoS Attack via Deep Learning," in *IEEE International Conference on Smart Computing (SMARTCOMP) 2017*, May 2017, pp. 1–8.

[8] L. Zhu, X. Tang, M. Shen, X. Du, and M. Guizani, "Privacy-Preserving DDoS Attack Detection Using Cross-Domain Traffic in Software Defined Networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 628–643, Mar. 2018.

[9] G. Sviridov, M. Bonola, A. Tulumello, P. Giaccone, A. Bianco, and G. Bianchi, "LODGE: LOcal Decisions on Global statEs in progrananaable data planes," in *IEEE Conference on Network Softwarization and Workshops (NetSoft) 2018*, June 2018, pp. 257–261.

[10] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with SDN data plane," in *IEEE Conference on Computer Communications (INFOCOM) 2017*, May 2017, pp. 1–9.

[11] . C. Lapolli, J. Adilson Marques, and L. P. Gaspary, "Offloading Real-time DDoS Attack Detection to Programmable Data Planes," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM) 2019*, Apr. 2019, pp. 19–27.

[12] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4 Edge node enabling stateful traffic engineering and cyber security," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 11, no. 1, pp. A84–A95, Jan. 2019.

[13] R. Jalili, F. Imani-Mehr, M. Amini, and H. R. Shahriari, *Detection of Distributed Denial of Service Attacks Using Statistical Pre-processor and Unsupervised Neural Networks*. Springer Berlin Heidelberg, 2005.

[14] "Spirent Test Center," https://www.spirent.com/products/testcenter, accessed Oct. 2019.

[15] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics New York, 2008.