Unleashing Dynamic Pipeline Reconfiguration of P4 Switches for Efficient Network Monitoring

Amir Al Sadi*, Marco Savi†, Andrea Melis*, Marco Prandini*, and Franco Callegati*
*Department of Computer Science and Engineering (DISI), University of Bologna, Bologna, Italy
†Department of Informatics, Systems and Communication (DISCo), University of Milano-Bicocca, Milano, Italy

***Abstract*—As it is happening in many fields that need efficient and effective classification of data, Machine Learning (ML) is becoming increasingly popular in network management and monitoring. We can say that ML algorithms are complex and, therefore better suited for execution in the centralized control plane of modern networks, but are also heavily reliant on data, that are necessarily collected in the data plane. The inevitable consequence is that the need to transfer lots of data from the data plane to the control plane may arise, with the risk of causing congestion on the control communication channel. Therefore it is of paramount importance to design systems capable of minimizing the interaction between data and control planes while ensuring good monitoring performance. The most recent generation of data plane programmable switches supporting the P4 language can help mitigate this problem by preprocessing traffic data at line rate. In this manuscript, we follow this approach and propose P4RTHENON: an architecture to distill in the data plane the relevant information to be mirrored to the control plane, where complex analysis can be performed. P4RTHENON leverages the P4-native support for runtime data plane pipeline reconfiguration to minimize the interaction between data and control planes while ensuring good monitoring performance. We tested our scheme on the volumetric DDoS detection use case: P4RTHENON reduces the volume of exchanged data by almost 75% compared to a pure control-plane-based solution, guarantees low memory consumption in the data plane, and does not degrade the overall DDoS detection capabilities.**

***Index Terms*—Programmable Data Planes, P4, Network Monitoring, Pipeline Reconfiguration, DDoS Detection**

I. Introduction

Networks are becoming by the day more pervasive in the processes of our daily lives, from work to leisure. This creates unimaginable opportunities but also opens the floor to new threats. For this reason, modern networks should promptly respond to unexpected events to safeguard the running services and avoid service disruption. Thus, to cope with legacy network technologies, modern infrastructures require in-depth and responsive network monitoring, management, and control.

Two key points, also relevant in the 5G world [1], are *(i)* control and data plane separation (the so-called CUPS approach) and *(ii)* network programmability. CUPS improves flexibility and scalability by de-coupling the logic problems from the pure data forwarding issues, while programmability allows networks to react to unwanted situations [2]. This envisions a *closed-loop approach* where the control plane collects real-time information about the status of the underlying network and reacts, by issuing suitable directives to the data plane, to modify its behavior [3, 4].

In this manuscript we present P4RTHENON, a viable approach to implement a closed-loop monitoring system, which can intercept network behaviors and take real-time actions. P4RTHENON stems from the idea of minimizing the congestion of the control channel between data and control planes [5], especially in the occurrence of abnormal behaviors. We validate our solution by devising a volumetric Distributed Denial of Service (DDoS) attack detection over P4RTHENON, to showcase how we can minimize the impact on the control channel and keep high detection rates. P4RTHENON follows the Software Defined Networking (SDN) paradigm: the detection logic is split between a simple data plane logic and a more complex control plane strategy. In this latter landscape, the goal of our approach is to achieve the best possible trade-off between monitoring performance, computational complexity, and control channel utilization.

To this aim, P4RTHENON splits the monitoring task into two phases called *coarse-grained* and *fine-grained* monitoring, i.e., *CGM* and *FGM*, respectively. The peculiarity of our approach is that FGM comes into place only when needed after an appropriate runtime *data plane pipeline reconfiguration*. For the sake of the proposed use case we implemented *(i) CGM_DDoS* strategy to detect the traffic portion suspected to belong to a DDoS attack, and *(ii) FGM_DDoS* strategy to deeply analyze the suspect traffic in the control plane and classify it in the right DDoS class if proven to be malicious.

We implemented CGM_DDoS as a simple in-network P4-based solution that calculates the degree of traffic asymmetry between two end hosts A and B in the two directions ($A \rightarrow B$ and $B \rightarrow A$), assuming that traffic is strongly asymmetric when a DDoS attack is in place, and flagging as suspect all the flows characterized by a high asymmetry degree. This strategy, based on a Count-min Sketch [6], over-estimates the number of DDoS attack flows, leading to some false positives while keeping the number of false negatives low.

Once suspect flows are identified by CGM_DDoS, P4RTHENON triggers FGM_DDoS *(i)* to extract relevant features from their packets in the data plane and *(ii)* to mirror this data to the control plane in the form of P4 digests. The collected features are then given as input to a trained Convolutional Neural Network (CNN), i.e., LUCID [7], performing ML inference and classifying any suspect flow as belonging to a DDoS attack class or as benign. According to our results, FGM_DDoS can substantially reduce false positives with respect to CGM_DDoS, thus achieving high Precision while keeping the control channel utilization low.

To summarize, the main contributions of this work are:

- A new architecture, i.e., P4RTHENON, to dynamically reconfigure the data plane pipeline at runtime and remodel the control plane accordingly, to minimize the control channel utilization.
- A lightweight P4 strategy to early detect Volumetric DDoS flows, i.e., Asymmetric Count-Min Sketch (ACMS).
- A validation of P4RTHENON to detect volumetric DDoS attacks, which analyzes the tradeoff between memory consumption/control channel occupation and the detection performance, leveraging a state-of-the-art ML agent [7] in the control plane.

The manuscript is organized as follows. We start by summarizing the state of the art on existing P4-based (i.e., in-network) and ML-based monitoring solutions (Section II), outlining their limitations. Section III details the principles our proposal, P4RTHENON, and its architectural components. Section IV presents the DDoS detection use case, from the scenario to the implementation of a testbed, and the performance of the proposed solution is evaluated in Section V. Finally, we draw our conclusions in Section VI.

II. Related Work

In this Section, we analyze the state of the art regarding SDN monitoring solutions. First, we sum up the existing programmable data plane-based monitoring solutions. Then, we investigate how ML-based monitoring is exploited in the SDN control plane. We proceed to give an overview of works that integrate data plane and ML-centered control plane solutions for monitoring. In anticipation of our architectural choice, we conclude the Section by analyzing the pipeline reconfiguration methods proposed in the literature.

A. Monitoring solutions with programmable data planes

Offloading part of the control plane intelligence to the data plane has become increasingly popular in SDN [8] thanks to the rise of data plane programmability (DPP). DPP opened the field to greater monitoring expressiveness on network devices since it can be leveraged to describe arbitrary, albeit simple packet manipulation strategies on top of regular forwarding. In recent years, programmable data planes have proven to be effective in supporting complex monitoring strategies by coding part of them directly on the data plane [9], most commonly exploiting the P4 language [10].

The most straightforward approach showing how DPP can be exploited to support network monitoring is *In-band Network Telemetry* (INT) [11], a framework proposed by some of the biggest networking companies in conjunction with the P4 Working Group. INT allows gathering monitoring information by transparently adding custom headers to users' packets, which are then extrapolated and forwarded to a centralized collector. INT has been extensively used to support traffic engineering [12], congestion control [13], and routing [14].

Another possibility to take advantage of DPP for network monitoring consists in exploiting the stateful memory made available by P4-based programmable data planes (i.e., P4 registers), to implement customized data structures (i.e., sketches [15]) for advanced *in-network monitoring* [16]. Thanks to these data structures, it is possible to support complex tasks, such as intrusion or anomaly detection, by keeping track of flows' state and aggregate statistics directly in the data plane. For instance, many strategies have been proposed to detect heavy flows (or *heavy hitters*) [17], using different data structures such as hash tables [18] or invertible sketches [19].

Recently, some works have also proposed to offload ML inference to the data plane, meaning that the whole ML model is made executable in the data plane pipeline in support of widely different monitoring tasks. For example, pForest [21] and BACKORDERS [22] have proposed to offload a random forest (RF) [30] on the data plane for in-network inference. These works either require a large amount of Match-Action Tables (MAT) installed or use arbitrary F1Score thresholds to rate the detection quality. A further step in this domain has been made by Taurus [20], which offloads CNN [31], Deep Neural Networks (DNN) [32], and Support Vector Machine (SVM) [33] models to the data plane exploiting MapReduce [34]. However, Taurus shows limitations already at model-level accuracy, which is below 70% for the DNN, while we could not find accuracy benchmarks for the other two models. Other works, such as [23, 24, 25], offload different ML models, among which Decision Trees (DT) and Binary Neural Networks (BNN), to the data plane. Especially, DT and BNN exhibit low inference performance. This is because the DT can only have limited depths and BNN adopts simple binary weights to overcome data plane operational limitations. Razavi et al. [25] implement a DNN, but the authors encoded weights' floating point numbers with 8-bit integers, leading to similar performance degradation as [23, 24]: this is a major limitation that they also clearly highlight in the paper.

Table I provides a brief summary of the aforementioned ML-based data plane solutions, all exploiting the P4 language technology, outlining their limitations. To summarize, trying to fit ML models to the programmable data plane is not simple, requires nontrivial operations to optimize the code and memory consumption, and high inference performance is hard to achieve. These are the reasons why P4RTHENON relies on simple sketch-based strategies in the data plane for CGM, while a ML-based in-depth analysis, as that performed by FGM (see Section II-B), leverages the more powerful control plane computational capabilities.

DDoS attack detection: As specified in the Introduction, in this paper we focus on DDoS attack detection as a use case. To address this problem, many solutions based on DPP have been proposed [35]. ML-based methods in the data plane have also been proposed, as BACKORDERS [22] already discussed above. Other works, e.g. [36, 37], adopt a coarse-grained strategy by definition, where the data plane is employed as a valid support to roughly detect anomalies. Ding et al. [38] propose INDDoS, a pragmatic way to detect victims targeted by a DDoS attack using Direct Bitmap combined with a Count-min Sketch. P4RTHENON's DDoS detection

Work	ML model	Limitation
Taurus [20]	CNN, DNN, SVM	Model accuracy for DNN below 70%
pForest [21]	RF	Detection quality evaluated using an arbitrary F1Score threshold
BACKORDERS [22]	RF	Requires a minimum of 63 MAT to work
Qin et al. [23]	BNN	Low Recall in certain datasets
Siracusano et al. [24]	DT	Low detection performance in certain datasets
Razavi et al [25]	DNN	8-bit floating point arithmetic

TABLE I: Limitations of current P4-based ML data plane solutions.

Work	ML model	Dataset	Number of features	True Positive Rate	Training Time on GPU
Ghanbari et al. [26]	CNN+SVM	CAIDA [27]	60	87.35%	N/A
DeepDefense [28]	CNN+RNN	UNB ISCX [29]	20	$\sim$98%, not clear if on unseen data	N/A, [7] claims $> 25\ h$
LUCID [7]	CNN	CAIDA [27], UNB ISCX [29], and other	12	$\sim$99% for each dataset	4500 s

TABLE II: Comparison between ML-based control plane DDoS detection solutions.

implementation indeed relates to other P4-based solutions mentioned in the AlSabeh et al. survey [35]. However, none of the considered schemes provides an in-depth analysis of the memory and control channel utilization when data and control planes interact while ensuring acceptable DDoS detection performance. Moreover, our P4 implementation of asymmetric flow detection is a novel contribution. We argue that our analysis not only validates the scalability of P4RTHENON, but also demonstrates how it is possible to match the detection performance of state-of-the-art solutions while drastically reducing the management overhead. In fact, our proposed CGM_DDoS strategy takes inspiration from the cited works, but it simplifies the strategy even further at the expense of increasing the false positive rate, which is then corrected by FGM_DDoS.

B. ML-based monitoring with SDN control planes

The effectiveness of ML-based solutions involving the SDN centralized control plane for monitoring has been thoroughly demonstrated [39]. The most important factors contributing to their success are the following [40]: *(i)* a single ML model can be deployed on top of the centralized controller to monitor network-wide scenarios; *(ii)* centralizing data collection is key for precise prediction; *(iii)* relevant data can be retrieved in real-time by the controller. In the following, we will specifically focus on monitoring tasks related to ML-based DDoS attack detection, from which we took inspiration to design FGM_DDoS.

DDoS attack detection: A thorough high-level analysis of ML techniques to detect DDoS attacks is proposed by He et al. [41], which outlines detection performance differences when selecting various features and models. This work also suggests that classic ML approaches are usually highly dependent on feature choice and datasets. To better generalize the model and loosen the constraint of selecting a fixed set of features, Deep Learning-based schemes have become extremely popular in detecting DDoS attacks. Among them, Ghanbari et al. [26] propose a solution that leverages a CNN, achieving high detection rates on a very well-known dataset, i.e., UNB ISCX intrusion detection evaluation dataset [29]. DeepDefense [28] is another example that combines a CNN and a Recurrent Neural Network (RNN), evaluated with good performance in the CAIDA DDoS 2007 attack dataset [27]. However, both solutions require a high number of features and do not suit real-time scenarios given their complexity. LUCID, proposed by Doriguzzi et al. [7], adopts similar concepts but in a way that makes the trained ML model suitable for online scenarios. LUCID is a lightweight CNN that classifies each traffic flow as belonging to a known DDoS class or as benign. With a rather fast training phase and a limited number of needed features, LUCID is capable of high detection rates.

Table II reports a comparison summary of the three works discussed above. LUCID ensures limited training time while keeping detection performance high, and thus it is a suitable solution to be adopted on top of an SDN control plane. However, it needs to inspect all the network traffic to provide good prediction rates. Mirroring packets to the control plane during a volumetric attack could congest the control channel [42, 43], in fact propagating it. Our proposed FGM_DDoS strategy exploits LUCID to classify network traffic, but it adds a data plane data aggregation logic to relieve the control channel, by delivering to the control plane, via P4 digests, only features extracted from suspect traffic in the data plane.

C. Interaction between data and ML-based control planes

Some works in literature have proposed monitoring architectures envisioning a tight interaction between programmable data and control planes in an SDN environment, with the goal of implementing refined strategies to optimize such an interaction: this is to some extent also the main objective of P4RTHENON. It is important to note that all the previous work on this topic focuses on anomaly/attack detection tasks.

Zhang et al. propose POSEIDON [44], a framework to map attack countermeasures to the programmable data plane and to servers. They propose a language for hardware abstraction and a runtime environment to orchestrate the real-time reaction to attacks. However, this solution requires multiple technologies and components and is bound to the language proposed by the authors, making it hardly replicable. A general solution that jointly maps ML-assisted detection in a programmable data plane and in a control plane is IIsy [45]. Two models are proposed: a lighter one, fully deployed in the data plane, and

a heavier one, in the control plane. They intensively tested the deployment of different models in the data plane, but do not consider the possibility of swapping between different configurations at runtime.

ORACLE [46] and the work proposed by Musumeci et al. [47] focus on two architectural approaches that envision a collaboration between control and programmable data plane to detect DDoS attacks. In both cases, aggregated statistics from packets' flows are computed by the programmable data plane and forwarded to the control plane, where they are processed by an ML engine to detect attacks through ML inference. Although simple and effective, these solutions require intense and constant communication between the data and the control plane, even when the attack is not happening, as they require the data plane to constantly send the aggregated statistics to the control plane. While taking inspiration from these proposals, our solution is the first attempt known to us to design a two-phase system for the optimization of control channel usage: in-network monitoring is autonomously performed by the programmable data plane to detect suspect traffic, and a more refined ML-based analysis takes place in the control plane. Differently from [46, 47], the latter is performed on categorical features extracted from packets belonging to suspect flows. Extracting packets' categorical features (e.g. IP flags) instead of computing flows aggregated statistics is another peculiarity of our proposal. Thanks to this, multi-class classification can be better performed (e.g. to which type of DDoS attack the packet belongs, if malicious) instead of only performing binary classification (DDoS/benign) as done in [46, 47].

FlowLens [48] is another work that resembles our solution regarding purpose and scope. Its authors propose an SDN architecture that leverages programmable switches to efficiently support multi-purpose ML-based security applications. FlowLens collects features related to packets distribution at line speed and classifies flows directly in the switches, using their CPU. However, though highly flexible and reliable, FlowLens cannot benefit from the network-wide view provided by a centralized SDN control plane. In addition, it does not envision any data plane pipeline reconfiguration at runtime, as supported by P4RTHENON. Reconfiguring the data plane pipeline makes it possible to install specialized pipelines, instead of using a general-purpose one, and to optimize the amount of data exchanged between data and control plane.

D. Programmable data planes pipeline reconfiguration

The potential advantages of runtime data plane pipeline reconfiguration have already attracted the attention of the research community. We argue that the work presented by Xing et al. [49] is the most convincing attempt to (re)program a switch at runtime. In this work, the authors propose an extension of the P4 language that enables partial reconfiguration of the data plane with minimum resource overhead, without service disruption, and guaranteeing consistent packet processing. By allowing developers to load new features at runtime on a reserved memory area, the authors propose a solution to the notorious problems of repopulating all the existing tables and of the introduced delay when a switch firmware is replaced. This work does not consider any specific application domain and it is not clear what the impact would be if the whole pipeline had to be reconfigured. A similar proposal was advanced by Feng et al. [50]. The authors designed a specific real-time upgradable architecture called In-situ Programmable Switch Architecture (IPSA). This approach allows to implement reconfiguration in a more efficient way, but at the cost of having to upgrade the whole network to adopt switches whose architecture follows the IPSA one, which may not be feasible in large-scale scenarios.

In contrast to the existing works, P4RTHENON leverages the native feature made available by the P4Runtime specification [51] that allows a P4 pipeline reconfiguration at runtime. We choose this approach because P4Runtime is a well-established Application Programming Interface (API) for controlling the data plane elements of a device whose behavior is specified by a P4 program, and thus no architectural change is needed as long as a P4-enabled device is adopted. To the best of our knowledge, the few experiments that we were able to find about partial or total pipeline reconfiguration have never focused on the optimization of the burden on the control channel. Conversely, P4RTHENON is specifically designed to minimize the amount of data exchanged between the involved planes, while ensuring high monitoring performance.

III. P4RTHENON: Monitoring Architecture

In this Section, we describe the main concepts behind P4RTHENON, our general-purpose real-time solution to describe and implement monitoring policies. The main goal of P4RTHENON is minimizing the amount of data exchanged between data and control planes while achieving high performance of monitoring engines running in the control plane that require detailed features extracted from packets, such as ML-based ones.

P4RTHENON executes monitoring tasks in two phases: *(i)* in the first phase, an approximate traffic analysis is performed, which identifies the flows that should be monitored more in depth (i.e., *Coarse-Grained Monitoring*); *(ii)* in the second phase, an accurate analysis is done on the flows selected by CGM, with the aim of further discriminating what flows meet the behavior specified by the monitoring policies (i.e., *Fine-Grained Monitoring*). Each phase is associated with a specific strategy deployed by the control plane, which requires a runtime reconfiguration of the data plane pipeline.

Specifically, CGM is meant to run completely in the data plane, meaning that just a few data points need to be forwarded to the control plane in this phase. Based on the information gathered during CGM, when meeting some pre-defined condition, the control plane triggers FGM, with a consequent reconfiguration of the data plane pipeline. FGM is data-intensive as it requires traffic features to be mirrored from the data plane to the control plane. However, the features mirrored during FGM are only those extracted from flows selected by CGM: this significantly reduces the amount of data to be forwarded and analyzed, enhancing Precision by lowering the input detection noise and reducing the burden on the control channel. Whenever some other pre-defined condition is met (e.g. after a timeout expiration), P4RTHENON

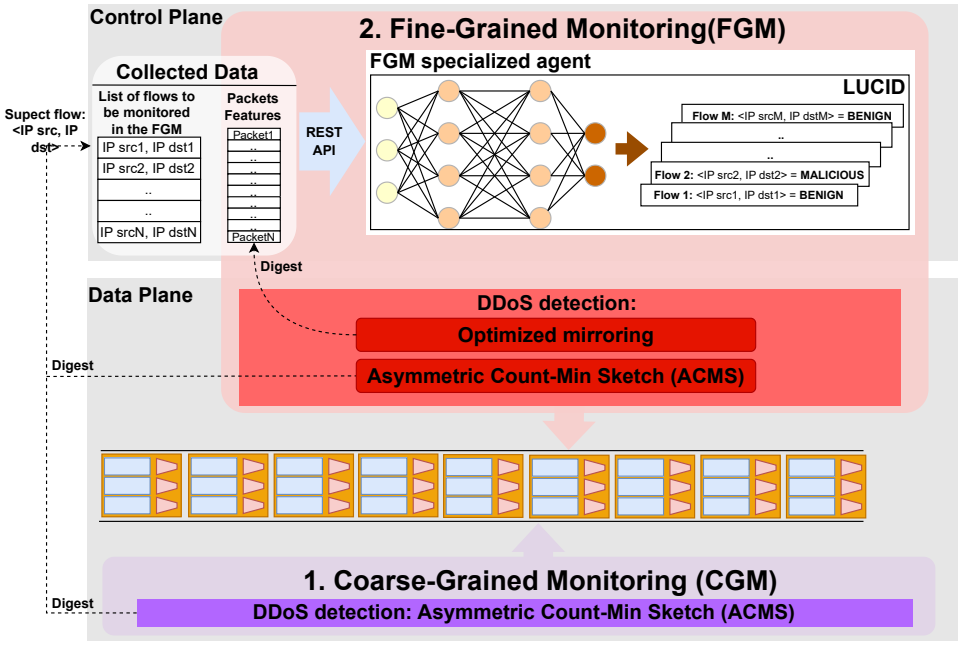


Fig. 1: P4RTHENON architecture with a focus on the DDoS detection use case (Section IV).

triggers the return to the execution of CGM and the pipeline is reconfigured to the previous status accordingly. Figure 1 illustrates the architecture of the system, also showing the specific strategies adopted in our DDoS detection use case, whose design and implementation will be detailed in Section IV. In the following we will provide further details on CGM and FGM, on design principles, and on the main enabling technologies.

A. Coarse-Grained Monitoring vs. Fine-Grained Monitoring

CGM is the default strategy installed in the data plane. It is designed to be executed on top of the regular forwarding with a low-added overhead. It relies on a very simple monitoring strategy that *(i)* continuously monitors the traffic; *(ii)* identifies the set of flows that meet some monitoring requirements. CGM is fully executed by the programmable data plane, and exchanges minimal data with the control plane. In fact, the data plane occasionally sends management messages to the control plane updating it with a summary of the current network status. Then, the control plane inspects the collected data and, if a condition is met, the execution of FGM is triggered.

FGM is instead deployed by the control plane whenever some traffic needs to be inspected with higher Precision. CGM is responsible for identifying the traffic flows worth being monitored by FGM. Unlike CGM, FGM's logic is evenly split between data and control planes. In the data plane, FGM extracts relevant features (e.g. IP flag, TCP ports, etc.) from packets of selected flows and mirrors them to the control plane. In the control plane, a specialized agent (e.g. a trained ML model) takes as input the extracted features and performs a deeper monitoring (e.g. flow classification).

B. Main design principles of P4RTHENON

CGM and FGM should be designed to ensure that CGM is able to recognize *all* (ideally) the flows that may need attention, but it could include in such a set also flows that are wrongly selected as interesting. Instead, FGM should be capable of further discriminating, from the set of flows selected by CGM, the flows that are truly relevant. The use case presented later in this manuscript (see Section IV), which refers to DDoS detection, will show that CGM_DDoS is effective in identifying a superset of flows that belong to DDoS attacks, hence finding all the true positives with a certain degree of false positives, while FGM_DDoS is very efficient at trimming out all the false positives. As far as these design principles are met, P4RTHENON could be adopted for widely different monitoring tasks other than DDoS detection.

C. Programming and interaction of the architectural elements

The data plane pipeline's behavior is specified by a program written in the *P4 language* [10]. Each P4-programmable pipeline consists of a set of processing blocks, which can modify the packet headers and gather packet-related data (e.g. the features required by FGM). As Southbound Interface (SBI) we adopt the well-known *P4Runtime* [51], which is exploited to *(i)* install match-action rules (enabling the selective per-flow features' mirroring in FGM) and *(ii)* send data to the control plane (e.g. extracted features) by means of *digest* messages.

The digest is a type of message specified in the P4Runtime specification [51] that can be adapted to send one-way data recovered by the data plane to the control plane. As the documentation explains, it differs from *packet-in* messages [52] as it is optimized to only send some packet's header fields and metadata, while packet-in is generally used to also send the payload. Multiple digests can be aggregated by P4Runtime

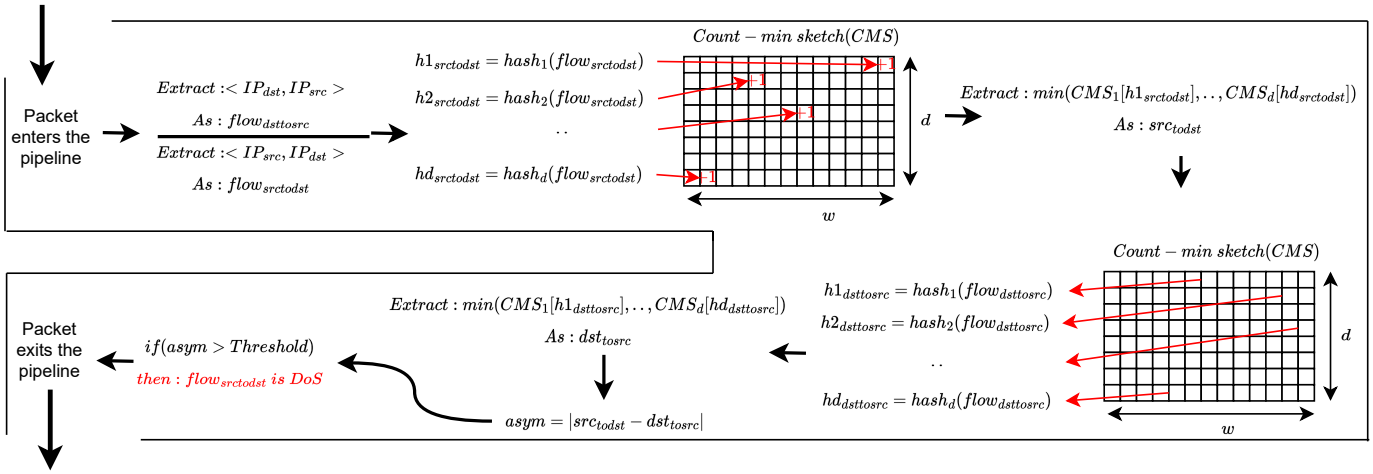


Fig. 2: Asymmetric Count-min Sketch description. A detailed explanation can be found in Section IV-B1.

into larger messages to reduce their number. The control plane retrieves the digest data as a *JSON collection*, where each JSON encapsulates a digest associated with a packet. The FGM specialized agent (see Fig. 1), which is implemented as a *Python script*, is continuously fed by the JSON collection, relying on a *RESTful* communication.

IV. P4RTHENON USE CASE: DDOS DETECTION

This Section illustrates the use case we chose to validate our approach. The produced code has been open-sourced[1]. We considered volumetric DDoS detection as an example to showcase P4RTHENON peculiarities. We will refer to the specialized versions of CGM and FGM as CGM_DDoS and FGM_DDoS, respectively. A preliminary investigation on the considered use case can be found in [53].

A. Asymmetric Count-min Sketch (ACMS)

To detect suspect DDoS attacks in the data plane, we devised a simple sketch-based algorithm implemented in P4 called *Asymmetric Count-min Sketch* (*ACMS*, see Fig. 2). ACMS was designed by observing the behavior of volumetric DDoS attacks, which usually generate a large number of packets toward the victim by means of a large number of compromised clients belonging to a botnet. In particular, ACMS is designed to detect flows with an unexpected asymmetry rate. In this condition, the traffic volume between the compromised client and the victim is expected to be much larger than the traffic volume in the opposite direction. Note however that P4RTHENON could be configured to support different types of attacks and multiple flavors of DDoS attacks, e.g. DDoS attacks that target a specific destination (as analyzed in [38]). ACMS incorporates two algorithms, i.e., *Count-min Sketch* and *asymmetric flow detection*:

[1]https://github.com/UniboSecurityResearch/p4runtime-go-client/tree/cnn-integration

1) Count-min Sketch (CMS) [6]: It exploits a probabilistic, low-memory data structure (i.e., sketch) that can be used to estimate *flows' packet count*, i.e., the number of packets carried by any network flow in a time window. It relies on two operations carried out on the sketch: *(i) Update*, to keep the count of incoming packets updated in the sketch; *(ii) Query*, to estimate the number of counted packets for a given flow. CMS relies on d different pairwise-independent hash functions, each with an output size w. The data structure is composed of a matrix of $d \cdot w$ counters: the packet-count estimation accuracy increases as the two dimensions increase, and vice versa, with theoretical bounds that have been proven [6].

2) Asymmetric flow detection: It is a simple in-network algorithm (proposed in P-SCOR [4]) that calculates whether a flow is part of a potential DDoS attack. It uses a fixed $Threshold$, a data structure called R that includes w counters, and a hashing function h that returns a number between 0 and $w-1$. Every time a packet crosses the switch, k is calculated as the hash of the $s =< IP_{src}, IP_{dst} >$ string, i.e., $h(s) = k$. The counter of R in the k-th position, i.e., $R(k)$, is then incremented ($R(k) = R(k)+1$). The algorithm then calculates $h(s') = j$, where $s' =< IP_{dst}, IP_{src} >$, and the asymmetry rate $asym = |R(k) - R(j)|$: if $asym > Threshold$, the flow is marked as a potential DDoS attack, as the difference of the traffic volume in the two directions is abnormal. The choice of identifying and tracking network flows considering the $< IP_{src}, IP_{dst} >$ couple rather than the more typical 5-tuple flow definition ($< IP_{src}, IP_{dst}, port_{src}, port_{dst}, protocol >$) has been made to ensure a slim approach in the data plane. Distinguishing 5-tuple malicious flows within $< IP_{src}, IP_{dst} >$ is a duty left to the control plane.

B. Strategies Description

1) CGM_DDoS: The strategy leverages ACMS as follows (Fig. 2). When a packet enters the data plane pipeline, the algorithm updates a CMS to increase the packets' counter for the considered flow. Then, the CMS is queried to retrieve the packet count estimation for the flow in the forward direction

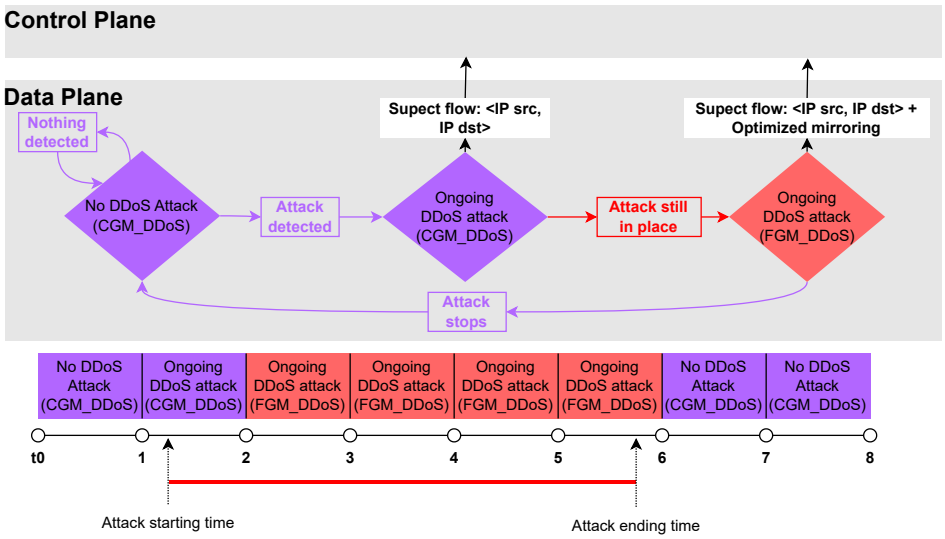


Fig. 3: Flow diagram of transitions between CGM_DDoS and FGM_DDoS (top) and timeline (bottom).

represented by the key $< IP_{src}, IP_{dst} >$, i.e., src_{todst}. The CMS is then queried again using the key $< IP_{dst}, IP_{src} >$ to retrieve the estimated packet count in the backward direction, i.e., dst_{tosrc}. The asymmetry rate is then computed as $asym = |src_{todst} - dst_{tosrc}|$: if $asym$ is higher than a value $Threshold$ the flow is labeled as *suspect*, and an alert is sent to the control plane in the form of a digest. The CMS is reset by the control plane every time a fixed *time window* expires.

It must be noted that setting the most appropriate $Threshold$ is not trivial and could affect the detection performance in both CGM_DDoS and FGM_DDoS. In Section V we will report the results of a sensitivity analysis aimed at determining what $Threshold$ best suits our scenario.

2) FGM_DDoS: It comes into place, through a data plane pipeline reconfiguration, following an alert that is sent to the control plane during CGM_DDoS. It includes both a *data plane* and a *control plane* logic.

The *data plane* logic in the P4-programmable pipeline combines two sub-strategies, namely *(i) ACMS* and *(ii) optimized mirroring*. ACMS is the same as that deployed in CGM_DDoS, and it is needed to keep monitoring any new suspect flow once the data plane pipeline has been reconfigured. Optimized mirroring is instead deployed to extract relevant features from packets and forward them to the control plane through digests. We call it *optimized* mirroring because it is meant to minimize the amount of data flowing on the control channel. It only mirrors features from packets belonging to flows deemed suspect by ACMS, both the ones marked as such during CGM_DDoS and, if any, those detected during FGM_DDoS. To further reduce the burden on the control channel, it also employs *packet sampling*, meaning that features from only $1\ out\ of\ N$ (i.e., *sampling rate* of $1/N$) suspect packets, flowing through the pipeline, are forwarded. N is a parameter that needs to be carefully set to strike the best balance between detection performance and control channel utilization, as we will show in Section V.

The *control plane* collects and stores the features extracted from the network traffic that are mirrored through the control channel. This data is then fed to a specialized online ML algorithm based on a pre-trained CNN model, i.e., LUCID [7], which pre-process it and performs a classification task to determine what suspect flows truly belong to a DDoS attack and what are instead benign. LUCID enriches CGM_DDoS analysis on $< IP_{src}, IP_{dst} >$ to further discriminate the 5-tuple malicious flows between source and destination.

C. Transition between CGM_DDoS and FGM_DDoS

The time is slotted in *time windows*, which starts at integer time reference $t_s = \bar{t}$ and lasts until $t_e = \bar{t} + 1$. At the beginning of each time window, it is possible to switch from CGM_DDoS to FGM_DDoS or vice-versa. Figure 3 shows an example of how the transition between the two strategies occurs. The top part of the figure reports a flow diagram showing state transitions in the face of a DDoS attack, while the bottom part focuses on a time perspective. Let's assume, as shown in the bottom part of the figure, that a DDoS attack starts during the second time window (in between $t_s = 1$ and $t_e = 2$) and expires in the sixth time window (in between $t_s = 5$ and $t_e = 6$). No other DDoS attack is in place in our time horizon, meaning that at the beginning of the first time window, CGM_DDoS is installed for coarse-grained traffic analysis.

During the first time window, nothing is detected by ACMS and no interaction between data and control plane occurs. During the second time window, as soon as the DDoS attack begins, CGM_DDoS starts sending alerts to the control plane every time a flow is considered to be suspect, as its asymmetry rate computed by ACMS overcomes the pre-defined threshold. After being notified of a possible attack, the control plane waits until the end of the current time window and then switches to FGM_DDoS, which requires a data plane pipeline reconfiguration: this happens at the beginning of the third time window, i.e., at $t_s = 2$. The reconfigured data plane starts

Algorithm 1 Asymmetric Count-min Sketch

Required: $packet, H, CMS, Threshold$
Optional output: $sendAlert$
$src \leftarrow packet.ip_{src}$
$dst \leftarrow packet.ip_{src}$
$CMS \leftarrow$ `updateCMS`(CMS, H, src, dst)
$min_{fwd} \leftarrow$ `queryCMS`(CMS, H, src, dst)
$min_{bwd} \leftarrow$ `queryCMS`(CMS, H, dst, src)
$asym \leftarrow |min_{fwd} - min_{bwd}|$
if $asym > Threshold$ **then**
  if $< src, dst > \notin suspect_{flows}$ **then**
    $suspect_{flows} \leftarrow < src, dst >$
    $sendAlert(src, dst)$
  end if
end if
return

function UPDATECMS(CMS, H, src, dst)
  $i \leftarrow 0$
  for all $hash \in H$ **do**
    $h \leftarrow hash(src, dst)$
    $CMS_i[h] \leftarrow CMS_i[h] + 1$ // row i
    $i \leftarrow i + 1$
  end for
  return CMS
end function

function QUERYCMS(CMS, H, src, dst)
  $i \leftarrow 0$
  $min \leftarrow \infty$
  for all $hash \in H$ **do**
    $h \leftarrow hash(src, dst)$
    if $CMS_i[h] < min$ **then** // row i
      $min \leftarrow CMS_i[h]$
    end if
    $i \leftarrow i + 1$
  end for
  return min
end function

extracting and mirroring features from (sampled) packets of the suspect flows identified by ACMS during CGM_DDoS, and at the same time monitors the rest of the traffic for potential new suspect flows. In the meantime, the control plane feeds the ML-based agent with packets' extracted features to identify malicious flows with high confidence. This condition holds until the DDoS attack ends, in this case during the sixth time window. As soon as this happens, the asymmetry rate of all flows falls behind the specified threshold, and at the beginning of the seventh time windows, CGM_DDoS can replace FGM_DDoS again.

D. Implementation

1) CGM_DDoS: To develop CGM_DDoS, we wrote ∼250 lines of P4 code. Our implementation of *Asymmetric Count-min Sketch* is summarized in Algorithm 1, including a description of the developed functions in P4. The P4 program specifies a CMS data structure as an array of P4 registers, which is used to summarize the number of packets per flow (i.e., packet count) in any direction. CMS is updated and queried leveraging a set of CRC32 hash functions (H), and the asymmetry $Threshold$ used to evaluate abnormal packet count differences in forward and backward flow directions is hard-coded in the program. Every time a $packet$ enters the P4 pipeline, the following operations are sequentially performed:

- `updateCMS`: the CMS is updated. The packet count for the $< ip_{src}, ip_{dst} >$ flow is increased by one unit. This is done by accessing, for each row i of the data structure, the cell with index equivalent to the hash value of $< ip_{src}, ip_{dst} >$, obtained by considering the i-th hash function from the set H, and increasing its value accordingly (see [6]).
- `queryCMS`: the operation is similar to the one illustrated in `updateCMS` but, instead of updating the value from the cell in each row i, the minimum among the stored values in the cells are kept to estimate the packet count for the corresponding flow (see [6]). `queryCMS` is executed twice, first to estimate the packet count for the forward flow $< ip_{src}, ip_{dst} >$, and then for the backward flow $< ip_{dst}, ip_{src} >$. Those values are called min_{fwd} and min_{bwd} respectively.
- The asymmetry rate ($asym$) is finally computed as $asym = |min_{fwd} - min_{bwd}|$ and if it exceeds the value $Threshold$, the $< ip_{src}, ip_{dst} >$ flow is considered suspect of belonging to a DDoS attack. When this happens, an alert is sent to the control plane in the form of a digest, which wraps 64 bits containing ip_{src} and ip_{dst} of the flow. To reduce the burden on the control channel, such an alert is generated only once per time window, at the first time that $< ip_{src}, ip_{dst} >$ leads to an $asym$ value greater than the $Threshold$.

Every Δt (time window size; in this paper we consider a $\Delta t = 30$ s) the switch sends a digest notifying the expiration of the window, which can result in two different outcomes: *(i)* if no flow is deemed suspect during the time slot, no action is required apart from resetting the counters of CMS; *(ii)* if at least one alert has been sent to the control plane during the window, the controller triggers FGM_DDoS.

2) FGM_DDoS: The P4-based *data plane* logic of FGM_DDoS is a superset of the logic of CGM_DDoS. In fact it includes ACMS (see Alg. 1) in its whole, with in addition:

1) A *feature extraction* logic to retrieve relevant features from packets flowing through the pipeline;
2) A *feature forwarding* logic to forward to the control plane only features *(i)* extracted from packets pertaining to suspect flows through ACMS and *(ii)* meeting the sampling requirements.

Together, 1) and 2) define the *optimized mirroring* strategy as described in Section IV-B. The feature extraction logic is detailed in Alg. 2, while the feature forwarding logic encapsulates the extracted metadata in a digest with a total size of 281 bits, which is sent to the control plane through the control channel using P4Runtime [51]. The procedure is shown in Alg. 3. The control plane then decodes the digest's data and saves the features in a JSON list.

The control plane exploits LUCID [7] for a finer-grained detection of DDoS attacks. LUCID includes a trained ML model (i.e., CNN) and a preprocessing algorithm, needed to reorganize retrieved features as required by the ML model (i.e., on a per-flow basis).

Feature	Description	Collected in	Protocol
ip_{src}	Source IP address of the packet	Parser	IPv4
ip_{dst}	Destination IP address of the packet	Parser	IPv4
ip_{flags}	IP flags used for fragmentation of the packet	Parser	IPv4
$ip_{protocol}$	Higher-layer protocol header encapsulated in the packet	Parser	IPv4
$ip_{totalLength}$	Size of the entire IP packet in bytes	Parser	IPv4
$icmp_{type}$	ICMP type of the ICMP packet	Parser	ICMP
udp_{len}	Length of the UDP segment in byte	Parser	UDP
tcp_{len}	Length of the TCP segment in byte	Parser	TCP
tcp_{ack}	Acknowledgement number of the TCP segment	Parser	TCP
tcp_{flags}	TCP flags of the segment (URG, ACK, PSH, RST, SYN, FIN)	Parser	TCP
$tcp_{srcPort}$	Source port number of the TCP connection	Parser	TCP
$tcp_{dstPort}$	Destination port number of the TCP connection	Parser	TCP
$tcp_{winSize}$	Window size of the TCP connection in bytes	Parser	TCP
$ingress_{timestamp}$	Timestamp of when packet is received in the ingress queue	Ingress Control Block	-

TABLE III: Packet features encapsulated in the digest sent to the control plane by optimized mirroring. The features in red are used by LUCID [7] in the preprocessing stage, but not for detection.

Algorithm 2 Optimized mirroring: Feature extraction

Required: $packet$
Output: $metadata$
  control block PARSER
    if $packet\ is\ IPv4$ **then**
      $metadata.ip_{(src,dst,flags,protocol,totalLength)} \leftarrow$
      $packet.ip_{(src,dst,flags,protocol,totalLength)}$
    end if
    if $packet\ is\ ICMP$ **then**
      $metadata.icmp_{type} \leftarrow packet.icmp_{type}$
    end if
    if $packet\ is\ UDP$ **then**
      $metadata.udp_{len} \leftarrow packet.udp_{len}$
    end if
    if $packet\ is\ TCP$ **then**
      $metadata.tcp_{(len,ack,flags,srcPort,dstPort,winSize)} \leftarrow$
      $packet.tcp_{(len,ack,flags,srcPort,dstPort,winSize)}$
    end if
    return $metadata$
  end control block

  control block INGRESS
    $metadata.timestamp \leftarrow ingress_{timestamp}$
    return $metadata$
  end control block

Algorithm 3 Optimized mirroring: Feature forwarding

Required: $metadata, N_{sampling}, suspect_{flows}$
Output: $sendDigest$
  $counter \leftarrow 0$
  if $< metadata.ip_{src}, metadata.ip_{dst} > \in suspect_{flows}$ **then**
    $counter \leftarrow counter + 1$
    if $counter == N_{sampling}$ **then**
      $sendDigest(metadata)$
      $counter \leftarrow 0$
    end if
  end if

For an online detection (i.e., classification) of malicious flows, the JSON list including the features is continuously sent to LUCID via RESTful communication. LUCID then aggregates and splits the traffic into flows, marking them as malicious or as benign by means of ML inference. The JSON list is emptied every Δt seconds, i.e., every time window expiration. This is done to reduce the amount of data stored in the control plane and to keep it updated on the current shape of the underlying traffic. If LUCID is fed with the most recent traffic, it is possible to spot whether a flow previously deemed as malicious starts behaving legitimately. In this case, the flow can be ruled out from the list of malicious flows.

V. PERFORMANCE EVALUATION

This Section presents a performance evaluation of P4RTHENON, with respect to the considered use case of DDoS detection, both from a resource consumption and detection capability point of view.

A. Evaluation metrics and methodology

The tests here presented are based on a labeled PCAP dataset (details in Section V-B), containing both true positives (TP, i.e., flows that belong to DDoS classes) and true negatives (TN, i.e., flows of benign traffic). The total number of flows is $T = TP + TN$. In each experiment, we obtain both false positives (FP, i.e., all the flows wrongly deemed belonging to a DDoS attack) and false negatives (FN, i.e., all those flows wrongly deemed benign). The *detection performance* is thus analysed by means of three metrics:

- $Precision = \frac{TP}{TP+FP}$. It measures how many of the positive predictions are correct. The higher the value, the lower the noise from false positives.
- $Recall = \frac{TP}{TP+FN}$. It measures how many positive cases are recognized. The higher the value, the lower the number of attacks escaping detection.
- $F1Score = \frac{2 \cdot Precision \cdot Recall}{Precision+Recall}$. It is computed as the harmonic mean of Precision and Recall, indicating an overall quality of the detection.

We also measure the average *Control Channel Utilization (CCU)*, which is defined as the amount of data transmitted on the control channel, which we call $collectedData_{size}$, in an observation time window Δt, i.e., $CCU = \frac{collectedData_{size}}{\Delta t}$. The higher CCU, the less efficient the strategy in terms of data-control plane interaction.

We divided our evaluation into four parts:

- *CGM_DDoS evaluation*, which presents a performance evaluation of our solution if only in-network data plane detection is performed. We compare it to an effective state-of-the-art in-network solution.

- *FGM_DDoS evaluation*, which analyses and validates our solution when ML-based control plane logic is installed. We evaluate the effectiveness and efficiency of the strategy over multiple combinations of ACMS thresholds and sampling rates.
- *Overall evaluation*, which summarises the results of CGM_DDoS and FGM_DDoS when combined, clearly pointing out the benefits of P4RTHENON with respect to other approaches.
- *Data plane pipeline reconfiguration evaluation*, which reports a brief discussion on the time needed by P4RTHENON to reconfigure the data plane pipeline while swapping between CGM_DDoS and FGM_DDoS.

Before delving into the obtained results, in the following, we give a concise description of the testbed and its settings.

B. Description of the testbed environment and parameters

Our experiments were carried out on in a virtual environment that consists of:

- An emulated single-switch network running on Mininet [54] with bmv2 [55] as P4 virtual switch target;
- A controller, developed in Go [56], responsible of *(i)* the information exchange with the data plane and *(ii)* reconfiguring the pipeline using the P4Runtime APIs.
- A process running LUCID, interacting with the controller via RESTful communication. LUCID was pre-trained using a dataset provided in its official repository[2], and model hyperparameters were set as the default ones specified in the paper [7]. For further details on LUCID's configuration the reader should refer to [7].
- A process simulating the DDoS attack by means of tcpreplay [57], which replays network traffic at 50 Mbps speed for a 6-minute long attack. We generated a PCAP sample dataset containing roughly 2 Gb of traffic. It is composed of 10% of benign traffic (taken from the CIC-IDS2017 dataset [58]) and 90% of DDoS traffic (generated with the hping3 [59] Linux utility). The attack speed is designed to saturate the switch, while the 6-minute duration allows replaying the dataset ~ 2 times. We generated traffic datasets with different numbers of malicious $< IP_{src}, IP_{dst} >$ flows (from 30 to 120) to stress the CMS with various traffic volumes: this aspect will be analyzed later in this Section.

All the components were executed on an Ubuntu 20.04 LTS Server with 14GB of RAM and 3 CPU cores KVM machine.

The in-network P4-based ACMS strategy uses a $(d = 2) \times (w = 1024)$ CMS, where every 48 bits are allocated to each cell, resulting in $2 \cdot 1024 \cdot 48 \sim 9.8Kb$ memory occupation. The two adopted hash functions are available by default in the bmv2's v1model.p4[3], i.e., $crc16$ and $crc32$.

C. CGM_DDoS evaluation

1) Sensitivity analysis of ACMS: To choose the right values d and w for CMS we conducted a detailed sensitivity analysis. Table IV shows a comparison between different values of d for $w = 1024$. It reports how the F1Score improves for $d = 2$ with respect to $d = 1$, and does not significantly improve further for $d = 3$ or more, meaning that $d = 2$ is a valid compromise between good detection performance and acceptable memory consumption. Figure 4 shows the orthogonal analysis, for fixed $d = 2$. Here, the Precision, Recall, and F1Score are collected over a variable *number of malicious flows*, with fixed w (Figure 4a) and fixed ACMS *thresholds* (Figure 4b). This analysis suggests that for $w = 1024$, the F1Score is significantly higher for any number of malicious flows compared with $w = 512$, and almost matches $w = 2048$. On the other hand, for $Threshold = 750$, the F1Score outperforms every other configuration. The threshold analysis anticipates the result we will further discuss in Section V-D. This investigation suggests that the best parameters for CGM_DDoS, given our settings, should be $d = 2$ and $w = 1024$. Moreover, in all the following experiments we will focus on a number of malicious flows equal to 60.

2) Comparison with the state of the art: We compare CGM_DDoS with an open-source[4], state-of-the-art solution called INDDoS [38]. As CGM_DDoS, INDDoS is an in-network P4-based solution that detects hosts targeted by volumetric DDoS attacks. It is threshold-based, like ACMS, i.e., the core strategy of CGM_DDoS: it estimates the *per-destination flow cardinality* (number of sources contacting a specific destination) and, if it is above a threshold value, the destination is considered under attack. To estimate it, *BACON sketch* is used: a data structure that combines a CMS and a Bitmap to *Update* and *Query* the per-destination flow cardinality once a packet enters the P4 pipeline. When the queried value crosses the specified threshold, a digest wrapping the *IP destination* of the victim is sent to the control plane.

The main difference between INDDoS and ACMS is that they focus on two different properties of volumetric DDoS attacks to detect them: the former on per-destination flow cardinality (which is expected to be high for destinations under attack), the latter on flows' asymmetry rate (which is expected to be high for malicious flows). We want to stress that INDDoS could replace ACMS as the core in-network algorithm of CGM_DDoS. However, if we look at Table V, some aspects can be highlighted. We decided to test CGM_DDoS and INDDoS considering their best configuration in terms of detection performance (F1Score) which are:

- INDDoS: $Threshold = 60$, BACON sketch of size $(d = 3) \times (w = 1024) \times (m = 1024)$ [38].
- CGM_DDoS: $Threshold = 750$, CMS of size $(d = 2) \times (w = 1024)$ (as shown in the previous subsection);

From Table V it can be seen that much more memory is used by INDDoS with respect to CGM_DDoS. In fact, the memory occupied by BACON sketch, considering that 1 bit is allocated to each cell [38], is $3 \cdot 1024 \cdot 1024 = 3145.7Kb$, which is more than 300 times higher than the memory occupied by the CMS adopted by CGM_DDoS (i.e., $9.8Kb$). However, INDDoS outperforms CGM_DDoS in terms of Precision, Recall, and

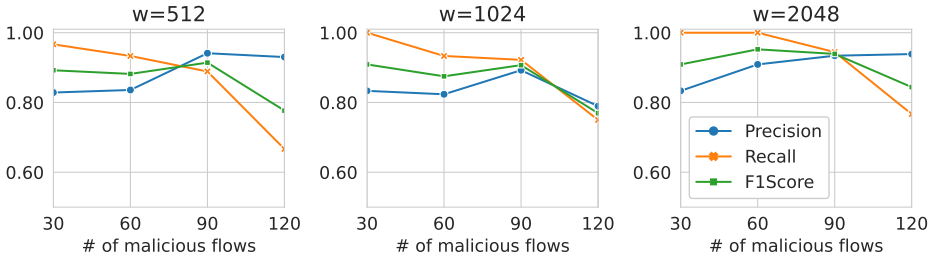
[2]https://github.com/doriguzzi/lucid-ddos/tree/master/sample-dataset

[3]https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4

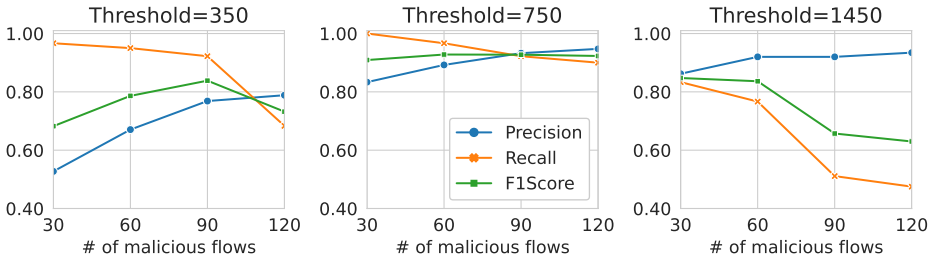
[4]https://github.com/DINGDAMU/INDDoS

Depth	Precision	Recall	F1Score	Memory in switch (Kb)
$d = 1$	0.54	0.95	0.69	4.9
$d = 2$	0.69	0.97	0.81	9.8
$d = 3$	0.71	0.97	0.82	14.7

TABLE IV: Detection comparison of ACMS while varying d (with $w = 1024$ fixed).



(a) ACMS variable width

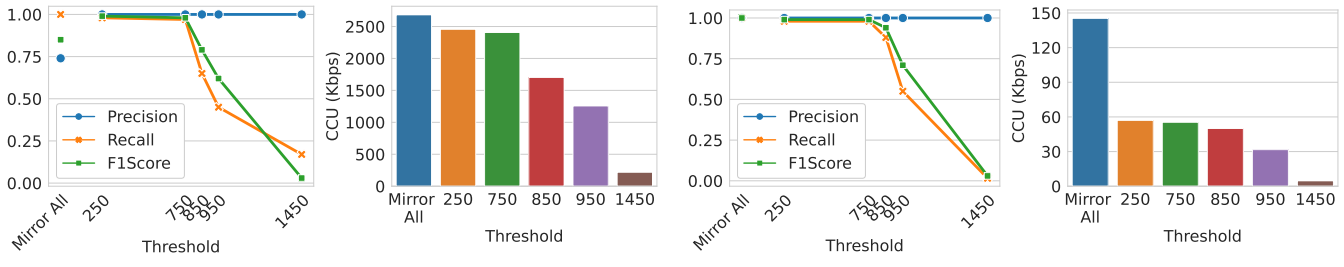


(b) ACMS variable threshold

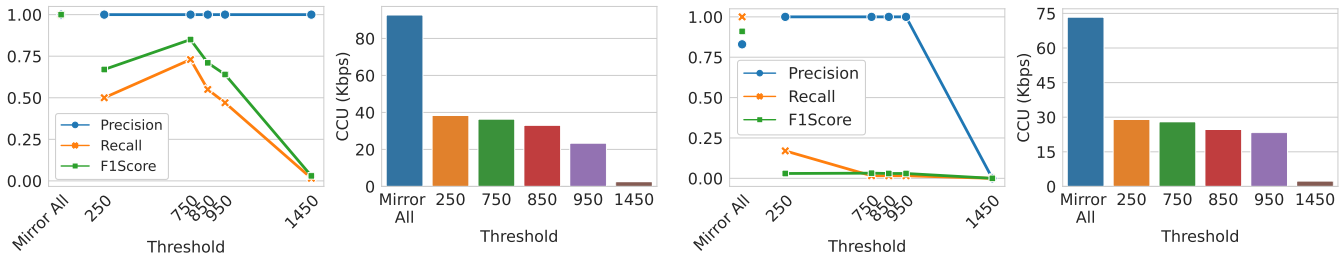
Fig. 4: CGM_DDoS: Detection performance for different ACMS widths (with $d = 2$ and $Threshold = 750$) and thresholds (with $d = 2$ and $w = 1024$) while varying the number of malicious flows.

Strategy	Precision	Recall	F1Score	CCU (Kbps)	Memory in switch (Kb)
InDDoS [38]	0.86	1	0.93	0.001	3145.7
CGM_DDoS (ACMS)	0.80	0.97	0.88	0.03	9.8

TABLE V: Detection comparison between an in-network state-of-the-art strategy [38] and CGM_DDoS.

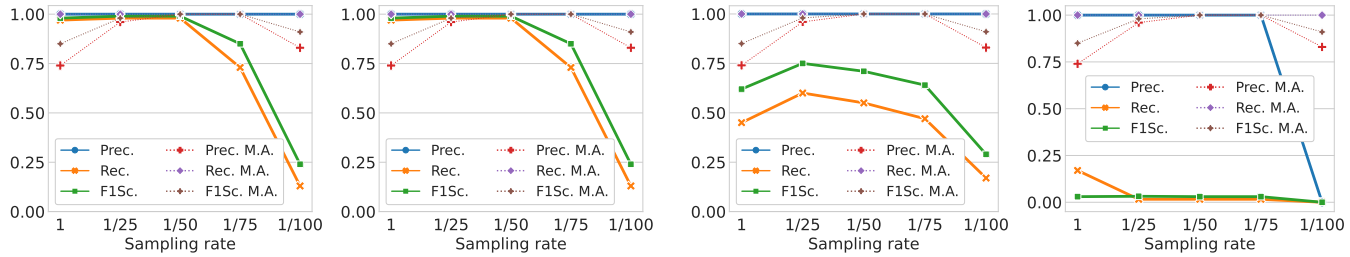


(a) $Sampling\ rate = 1$ (b) $Sampling\ rate = 1/50$



(c) $Sampling\ rate = 1/75$ (d) $Sampling\ rate = 1/100$

Fig. 5: FGM_DDoS vs. Mirror All: Detection performance and Control Channel Utilization for different thresholds (sampling rate fixed).



(a) $Threshold = 750$ (b) $Threshold = 850$ (c) $Threshold = 950$ (d) $Threshold = 1450$

Fig. 6: FGM_DDoS vs. Mirror All: Detection performance for different sampling rates (threshold fixed). M.A. = Mirror All, Prec. = Precision, Rec. = Recall, F1Sc. = F1Score.

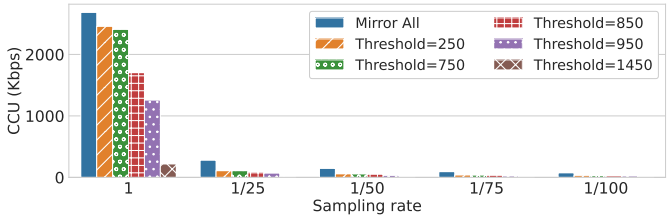


Fig. 7: FGM_DDoS vs. Mirror All: Control Channel Utilization for different sampling rates (threshold fixed).

F1Score. This is explained by the higher complexity (and required memory) of INDDoS compared with CGM_DDoS, which makes it a more performing stand-alone solution. However, Recall of both solutions is high (1 or close to 1), while Precision of both is just decent, with slightly worse performance for CGM_DDoS.

In addition, CCU of CGM_DDoS, although only in the order of tens of bps, is higher than CCU of INDDoS. This happens for two reasons: *(i)* the higher frequency of sent alerts, as CGM_DDoS sends an alert every time it spots a suspect flow, while INDDoS groups alerts by destination; *(ii)* the higher size of the digests payload, as CGM_DDoS sends 8-bytes alerts (IP source and IP destination of the flow), while INDDoS sends only 4-bytes alerts (IP address of the victim). In our testbed, the size of a CGM_DDoS alert is around $100\ bits$ after being encapsulated in a JSON structure, while the size of an INDDoS alert is around $50\ bits$.

To summarize, INDDoS provides a superior detection performance compared to CGM_DDoS. However, as specified in Section III-B, P4RTHENON requires a low number of false negatives for CGM, which is guaranteed by both strategies (high Recall), while it is tolerant to false positives, which can be filtered out by FGM. So, although CGM_DDoS Precision is slightly lower and CGM_DDoS CCU higher (but still low in absolute terms), its adoption in the place of INDDoS is fully justified by its much lower memory usage.

D. FGM_DDoS evaluation

In this Section, we analyze the benefits of FGM_DDoS. We provide an overview of the configurations we tested in the environment described in Section V-B, setting different ACMS *thresholds* and different *sampling rates*. The goal is to explore the existing trade-offs between detection performance (in terms of Recall, Precision, F1Score) and Control Channel Utilization, as these two configuration parameters are the most impactful on the above-mentioned metrics. Furthermore, we compare these results with a naïve strategy that we call *Mirror All* and is inspired by [43]: it does not provide ACMS-aided optimized mirroring, but it simply performs features extraction and forwarding from *any* packet, regardless of that it belongs to a suspect flow or not. In other words, it does not embed any ACMS logic and discriminating between benign and malicious flows is fully enforced by the control plane. As for FGM_DDoS, it is possible to reduce the burden on the control channel through sampling, i.e., by only forwarding features extracted from one packet out of N.

We tested different combinations of ACMS thresholds and sampling rates: in the following we report only the most significant combinations for the sake of conciseness. Figures 5 and 6 report the results for our tests, where for each configuration Precision, Recall, F1Score and CCU are reported. With respect to CCU reported values, we want to stress that in FGM_DDoS the size of a digest, including the packet's features, is around $2Kb$, i.e., 20 times the size of the CGM_DDoS one. Moreover, in CGM_DDoS a much lower number of digests is sent to the control plane, as only one digest per suspect flow, in any time window, is forwarded to the control plane. This is the reason why CCU for FGM_DDoS is several orders of magnitude higher than for CGM_DDoS (as reported in Table V).

Figure 5 reports the detection performance and CCU under four chosen sampling rates, namely, 1, $1/50$, $1/75$, $1/100$, and varying the ACMS threshold. Note that Mirror All is insensitive to the threshold as ACMS is not adopted, and thus in the left-hand-side subfigures its Precision, Recall and F1Score values are reported as single points. We can see that by increasing the threshold a negative impact on Recall is experienced, as the number of false negatives significantly increases. In fact, only flows with very high asymmetry rates are deemed suspect by ACMS and thus some malicious flows, with lower asymmetry rate, are neglected by ACMS. On the other hand, choosing a higher threshold has a very good impact on CCU, as features extracted by packets belonging to fewer flows (i.e., only those suspect) need to be forwarded to the control plane. Instead, Precision is not strongly affected and is always high, meaning that LUCID has a very good ability to filter out false positives.

Figure 5 also shows that lowering the sampling rate from 1 to $1/50$ is beneficial for both detection performance and Control Channel Utilization for high thresholds. CCU is lowered by one order of magnitude, while the detection performance (in terms of F1Score) increases. This phenomenon may seem counter-intuitive, however, by lowering the amount of data sent to the control plane, congestion on the control channel is reduced with consequent benefits on detection performance. In fact, congestion causes uncontrolled digests' discard, meaning that lower congestion reduces the amount of noise (in terms of flows' patterns alteration) given as input to LUCID. By further decreasing the sampling rate, e.g. $1/75$ and $1/100$, the number of packets' features sent to the control plane decreases up to a point that LUCID has not enough data to perform a proper classification. CCU is low but Precision, Recall, and F1Score are also low regardless of the threshold value.

The same trend is confirmed by looking at Figs. 6 and 7, which report the detection performance (Fig. 6) and CCU (Fig. 7) under four chosen value of the ACMS threshold, namely 750, 850, 950, 1450, and while varying the sampling rate. Fig. 6 shows that the detection performance peeks for sampling rates higher than $1/50$. However, the most important trend is clearly highlighted in Fig. 7: whenever sampling is performed, CCU for both Mirror All and FGM_DDoS drops significantly. For very low sampling rates ($< 1/75$) the same considerations as those done for Fig. 5, with respect to high thresholds, apply: in these cases, the amount of informative data sent to the control plane is too limited to ensure robust detection performance. Also the case for threshold values of 750 and 850 is interesting. With respect to detection performance (Fig. 6) they behave the same for any sampling rate, but CCU (Fig. 7) is reduced by 30% in the case of a threshold of 850.

By comparing FGM_DDoS with Mirror All, we can see that Mirror All performs best for sampling rates of $1/50$ and $1/75$. Its counter-intuitive worse detection performance with a sampling rate of 1 is due to the high congestion on the control channel. However, in all the cases, Mirror All leads to a much higher CCU than FGM_DDoS. Specifically, the same detection performance of Mirror All can be obtained by FGM_DDoS with a sampling rate $1/50$ and threshold of 750, while reducing CCU by around three times.

In summary, our results show that by choosing the most appropriate sampling rate and ACMS threshold our strategy makes it possible to find a good balance between detection performance and amount of traffic on the control channel.

E. Overall evaluation

Table VI summarizes the results obtained in Sections V-C and V-D, with respect to the following strategies and related configuration parameters:

- CGM_DDoS: $Threshold = 750$, CMS of size $(d = 2) \times (w = 1024)$;
- INDDoS: $Threshold = 60$, BACON Sketch of size $(d = 3) \times (w = 1024) \times (m = 1024)$ [38].
- FGM_DDoS: $Threshold = 750$, $Sampling\ rate = 1/50$, CMS of size $(d = 2) \times (w = 1024)$.
- Mirror All: $Sampling\ rate = 1/75$.

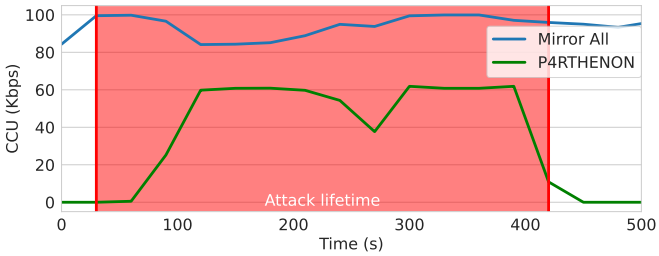


Fig. 8: Control Channel Utilization over time between P4RTHENON and Mirror All in their best configurations.

- P4RTHENON: $Threshold = 750$, $Sampling\ rate = 1/50$, CMS of size $(d = 2) \times (w = 1024)$.

P4RTHENON combines CGM_DDoS and FGM_DDoS via data plane pipeline reconfiguration, as specified in Sections III and IV. The parameters of each strategy has been chosen to maximize the detection performance (in terms of F1Score) as first objective and, in the case of multiple settings with the same detection performance, the one that minimizes CCU.

Table VI shows how P4RTHENON, FGM_DDoS and Mirror All outperform in terms of F1Score the in-network strategies that are fully executed in the data plane (i.e., CGM_DDoS and INDDoS). However, Recall is always high, meaning that all strategies, also those fully executed in the data plane, are good at effectively identifying true positives (i.e., the malicious traffic). It follows that the strategies relying on LUCID as an ML engine in the control plane (i.e., FGM_DDoS, Mirror All and P4RTHENON) have a much higher Precision, meaning that by deeply analyzing in the control plane the traffic features extracted from packets is very effective it to keep the number of false positives low. In the case of FGM_DDoS and P4RTHENON this property can be effectively exploited to filter out in the control plane the flows that are identified as suspect by ACMS in the data plane, which instead are benign. Aside from detection quality considerations, resource allocation and utilization are the aspects that make our proposed solution stands out. If we analyze CCU we can see that the in-network strategies lead to minimal usage of the control channel, while the others, for which feature extraction and forwarding to the control plane is needed, pay the price of a much higher average channel occupation. However, FGM_DDoS and especially P4RTHENON have a reduced CCU with respect to Mirror All, of around 40% and 75% respectively, as they benefit from the presence of ACMS to only forward features from suspect flows. P4RTHENON reduces CCU even further by having almost no interaction between control and data plane when CGM_DDoS is installed and attacks are not under stricter scrutiny.

Moreover, by looking at the occupied memory in the switch, we can stress again how INDDoS allocates a much higher amount of memory ($3145.7Kb$) than ACMS ($9.8Kb$), which is used in CGM_DDoS, FGM_DDoS and P4RTHENON. Instead, Mirror All does not require any data structure in the data plane, so it does not consume memory. This, however, comes at the expense of a significantly higher CCU.

Finally, a comprehensive look at Table VI shows how

Strategy	Precision	Recall	F1Score	Control Channel Utilization (Kbps)	Memory in switch (Kb)
CGM_DDoS (ACMS)	0.80	0.97	0.88	0.03	9.8
InDDoS [38]	0.86	1	0.93	0.001	3145.7
FGM_DDoS (ACMS + Optimized Mirroring)	1	0.98	0.99	55.3	9.8
Mirror All	1	1	1	92.7	0
P4RTHENON (CGM_DDoS + FGM_DDoS)	1	0.98	0.99	22.7	9.8

TABLE VI: Overall comparison between the different strategies.

P4RTHENON, thanks to its peculiarities, strikes the best balance between detection performance, CCU, and memory occupation with respect to the other strategies. Figure 8 confirms the speculation we drew from Table VI and reports a comparison between the two strategies with the best detection performance, i.e., Mirror All and P4RTHENON, in terms of CCU overtime during an attack, which is marked by a red area. The attack starts at $\bar{t} = 30s$: for P4RTHENON, CGM_DDoS is in place before this time instant, and a negligible amount of data is sent on the control channel. After $\bar{t}$, CGM_DDoS starts identifying suspect flows and after another Δt, at $t' = 60s$, FGM_DDoS is installed and optimized mirroring starts (correspondingly, CCU increases). Then, the attack ends at $t'' = 420s$ and, in the next time window, CGM_DDoS is restored and CCU drops to almost zero. By looking instead at Mirror All, we can see an almost constant CCU of around 90 Kbps as the features are extracted and forwarded from *any* packet, also when no attack is happening. Moreover, when the attack is in place, the data plane logic adopted by P4RTHENON (i.e., ACMS) makes it possible to save much control channel bandwidth by only forwarding features from suspect flows.

F. Data plane pipeline reconfiguration evaluation

We performed some experiments to evaluate the *system downtime* when a real-time P4 pipeline reconfiguration is performed to swap between CGM_DDoS and FGM_DDoS. It is important to stress that such an evaluation is strongly dependent on the adopted emulated environment and software switch target, and further tests will be performed as future work on hardware testbeds to confirm our findings. In our experiment we swapped between CGM_DDoS and FGM_DDoS 100 times and we measured the downtime during each transition, then calculating mean and variance. The computed mean is 263.4 ms, with a very low variance (0.6). Reconfiguring the pipeline, at least on Mininet with bmv2, is quick and stable.

VI. Conclusion

Minimizing data exchanged on the control channel for data-driven monitoring tasks is pivotal in complex networks. In fact, introducing a new feature or service should not be detrimental to the system. P4RTHENON is a scheme that supports the employment of lightweight and precise monitoring tasks to meet these requirements. It leverages P4-assisted real-time reconfiguration of programmable network devices, with minimal overhead and traffic loss.

We demonstrate the validity of our scheme by formulating a P4RTHENON-assisted solution to detect volumetric DDoS attacks. This strategy leverages two phases: *(i)* a pre-filtering stage to select the important portions of suspect traffic to analyze, and *(ii)* a fine-grained strategy that leverages optimized packet features' mirroring from the data plane towards the control plane, where a ML-based specialized agent attests what portion of suspect traffic is indeed malicious. This use case shows how P4RTHENON can reduce the cross-plane communication overhead by almost 80% while keeping high DDoS detection rates.

Being the use case is of practical significance, we foresee to proceed in its improvement by investigating its performance on a hardware testbed and by automating parameters' optimization (i.e., ACMS threshold and sampling rate) according to the traffic shape. In addition, we believe that the presented approach could be applied with profit to other, more complex network monitoring scenarios; our line of research will be correspondingly widened to encompass other use cases for a broader validation of P4RTHENON.

Acknowledgment

The research leading to these results has been partially funded by the Italian Ministry of University and Research (MUR) under the PRIN 2022 PNRR framework (EU Contribution – NextGenerationEU – M. 4,C. 2, I. 1.1), SHIELDED project, ID P2022ZWS82.

References

[1] Architecture Working Group. *View on 5G Architecture*. Tech. rep. Available on-line at https://5g-ppp.eu/wp-content/uploads/2014/02/5G-PPP-5G-Architecture-WP-July-2016.pdf. 5G PPP, 2016.

[2] F. Callegati et al. "SDN for dynamic NFV deployment". In: *IEEE Communications Magazine* 54.10 (2016), pp. 89–95. DOI: 10.1109/MCOM.2016.7588275.

[3] Davide Borsatti et al. "Mission Critical Communications Support With 5G and Network Slicing". In: *IEEE Transactions on Network and Service Management* 20.1 (2023), pp. 595–607. DOI: 10.1109/TNSM.2022.3208657.

[4] Andrea Melis et al. "P-SCOR: Integration of constraint programming orchestration and programmable data plane". In: *IEEE Transactions on Network and Service Management* 18.1 (2020), pp. 402–414.

[5] Menghao Zhang et al. "Control plane reflection attacks in SDNs: New attacks and countermeasures". In: *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21*. Springer. 2018, pp. 161–183.

[6] Graham Cormode and S. Muthukrishnan. "An improved data stream summary: the count-min sketch and its applications". In: *Journal of Algorithms* 55.1 (2005), pp. 58–75. ISSN: 0196-6774. DOI: https://doi.org/10.1016/j.jalgor.2003.12.001.

[7] R. Doriguzzi-Corin et al. "Lucid: A Practical, Lightweight Deep Learning Solution for DDoS Attack Detection". In: *IEEE Transactions on Network and Service Management* 17.2 (2020), pp. 876–889. DOI: 10.1109/TNSM.2020.2971776.

[8] Athanasios Liatifis et al. "Advancing sdn from openflow to p4: A survey". In: *ACM Computing Surveys* 55.9 (2023), pp. 1–37.

[9] Damu Ding et al. "Design and Development of Network Monitoring Strategies in P4-enabled Programmable Switches". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 2022.

[10] Pat Bosshart et al. "P4: Programming Protocol-Independent Packet Processors". In: 44.3 (July 2014), pp. 87–95. ISSN: 0146-4833. DOI: 10.1145/2656877.2656890. URL: https://doi.org/10.1145/2656877.2656890.

[11] Lizhuang Tan et al. "In-band network telemetry: A survey". In: *Computer Networks* 186 (2021), p. 107763.

[12] Vimalkumar Jeyakumar et al. "Millions of little minions: Using packets for low latency network programming and visibility". In: *ACM SIGCOMM Computer Communication Review* 44.4 (2014), pp. 3–14.

[13] Yuliang Li et al. "HPCC: High precision congestion control". In: *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 44–58.

[14] Naga Katta et al. "Clove: Congestion-aware load balancing at the virtual edge". In: *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*. 2017, pp. 323–335.

[15] Hui Han et al. "Applications of sketches in network traffic measurement: A survey". In: *Information Fusion* 82 (2022), pp. 58–85.

[16] Tooska Dargahi et al. "A survey on the security of stateful SDN data planes". In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1701–1725.

[17] Ran Ben-Basat et al. "Heavy hitters in streams and sliding windows". In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9.

[18] Ran Ben-Basat et al. "Efficient measurement on programmable switches using probabilistic recirculation". In: *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE. 2018, pp. 313–323.

[19] Lu Tang, Qun Huang, and Patrick PC Lee. "A fast and compact invertible sketch for network-wide heavy flow detection". In: *IEEE/ACM Transactions on Networking* 28.5 (2020), pp. 2350–2363.

[20] Tushar Swamy et al. "Taurus: a data plane architecture for per-packet ML". In: *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 2022, pp. 1099–1114.

[21] Coralie Busse-Grawitz et al. "pforest: In-network inference with random forests". In: *arXiv preprint arXiv:1909.05680* (2019).

[22] Bruno Coelho and Alberto Schaeffer-Filho. "BACKORDERS: using random forests to detect DDoS attacks in programmable data planes". In: *Proceedings of the 5th International Workshop on P4 in Europe*. 2022, pp. 1–7.

[23] Qiaofeng Qin et al. "Line-speed and scalable intrusion detection at the network edge via federated learning". In: *2020 IFIP Networking Conference (Networking)*. IEEE. 2020, pp. 352–360.

[24] Giuseppe Siracusano et al. "Re-architecting traffic analysis with neural network interface cards". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 2022, pp. 513–533.

[25] Kamran Razavi et al. "Distributed DNN serving in the network data plane". In: *Proceedings of the 5th International Workshop on P4 in Europe*. 2022, pp. 67–70.

[26] Maryam Ghanbari and Witold Kinsner. "Extracting features from both the input and the output of a convolutional neural network to detect distributed denial of service attacks". In: *2018 IEEE 17th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. IEEE. 2018, pp. 138–144.

[27] *DDoS 2007 attack*. https://catalog.caida.org/dataset/ddos_attack_2007. Accessed: 2023-6-15. DOI: https://catalog.caida.org/dataset/ddos_attack_2007.

[28] Xiaoyong Yuan, Chuanhuang Li, and Xiaolin Li. "DeepDefense: identifying DDoS attack via deep learning". In: *2017 IEEE international conference on smart computing (SMARTCOMP)*. IEEE. 2017, pp. 1–8.

[29] Ali Shiravi et al. "Toward developing a systematic approach to generate benchmark datasets for intrusion detection". In: *computers & security* 31.3 (2012), pp. 357–374.

[30] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[31] Keiron O'Shea and Ryan Nash. "An introduction to convolutional neural networks". In: *arXiv preprint arXiv:1511.08458* (2015).

[32] Wojciech Samek et al. "Explaining deep neural networks and beyond: A review of methods and applications". In: *Proceedings of the IEEE* 109.3 (2021), pp. 247–278.

[33] Shan Suthaharan and Shan Suthaharan. "Support vector machine". In: *Machine learning models and algorithms for big data classification: thinking with examples for effective learning* (2016), pp. 207–235.

[34] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113.

[35] Ali AlSabeh et al. "A survey on security applications of P4 programmable switches and a STRIDE-based vulnerability assessment". In: *Computer Networks* 207 (2022), p. 108800.

[36] Damu Ding, Marco Savi, and Domenico Siracusa. "Tracking normalized network traffic entropy to detect DDoS attacks in P4". In: *IEEE Transactions on Dependable and Secure Computing* 19.6 (2021), pp. 4019–4031.

[37] Alexandre da Silveira Ilha et al. "Euclid: A fully in-network, P4-based approach for real-time DDoS attack detection and mitigation". In: *IEEE Transactions on Network and Service Management* 18.3 (2020), pp. 3121–3139.

[38] Damu Ding et al. "In-network volumetric DDoS victim identification using programmable commodity switches". In: *IEEE Transactions on Network and Service Management* 18.2 (2021), pp. 1191–1202.

[39] Yanling Zhao et al. "A survey of networking applications applying the software defined networking concept based on machine learning". In: *IEEE Access* 7 (2019), pp. 95397–95417.

[40] Junfeng Xie et al. "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges". In: *IEEE Communications Surveys & Tutorials* 21.1 (2018), pp. 393–430.

[41] Zecheng He, Tianwei Zhang, and Ruby B Lee. "Machine learning based DDoS attack detection from source side in cloud". In: *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*. IEEE. 2017, pp. 114–120.

[42] Seungbeom Song et al. "A congestion avoidance algorithm in SDN environment". In: *2016 International Conference on Information Networking (ICOIN)*. IEEE. 2016, pp. 420–423.

[43] Roberto Doriguzzi-Corin et al. "Introducing packet-level analysis in programmable data planes to advance Network Intrusion Detection". In: *Computer Networks* 239 (2024), p. 110162.

[44] Menghao Zhang et al. "Poseidon: Mitigating volumetric ddos attacks with programmable switches". In: *the 27th Network and Distributed System Security Symposium (NDSS 2020)*. 2020.

[45] Changgang Zheng et al. "IIsy: Practical in-network classification". In: *arXiv preprint arXiv:2205.08243* (2022).

[46] Sebastian Gomez Macias, Luciano Paschoal Gaspary, and Juan Felipe Botero. "Oracle: An architecture for collaboration of data and control planes to detect ddos attacks". In: *2021*

IFIP/IEEE International Symposium on Integrated Network Management (IM). IEEE. 2021, pp. 962–967.

[47] Francesco Musumeci et al. "Machine-learning-enabled DDoS attacks detection in P4 programmable networks". In: *Journal of Network and Systems Management* 30 (2022), pp. 1–27.

[48] Diogo Barradas et al. "FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications." In: *NDSS*. 2021.

[49] Jiarong Xing et al. "Runtime programmable switches". In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 2022, pp. 651–665.

[50] Yong Feng et al. "In-Situ Programmable Switching Using RP4: Towards Runtime Data Plane Programmability". In: *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*. HotNets '21. Virtual Event, United Kingdom: Association for Computing Machinery, 2021, pp. 69–76. ISBN: 9781450390873. DOI: 10.1145/3484266.3487367. URL: https://doi.org/10.1145/3484266.3487367.

[51] The P4 Language Consortium. *P4Runtime Specification*. 2020. URL: https://p4.org/p4-spec/p4runtime/v1.3.0/P4Runtime-Spec.pdf.

[52] *OpenFlow Switch Specification, Version 1.5.1 (Protocol version 0x06)*. Mar. 2015. URL: https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf.

[53] Amir Al Sadi et al. "Real-time Pipeline Reconfiguration of P4 Programmable Switches to Efficiently Detect and Mitigate DDoS Attacks". In: *2023 26th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. 2023.

[54] Karamjeet Kaur, Japinder Singh, and Navtej Singh Ghumman. "Mininet as software defined networking testing platform". In: *International conference on communication, computing & systems (ICCCS)*. 2014, pp. 139–42.

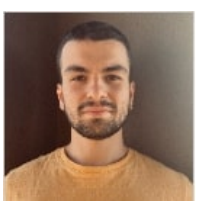
[55] P4 Language Consortium et al. *p4lang/behavioral-model*. 2019.

[56] Google. *The Go programming language*. 2023.

[57] Linux foundation. *tcpreplay*. https://linux.die.net/man/1/tcpreplay. 2023.

[58] "CICIDS2017 Dataset". In: *https://www.unb.ca/cic/datasets/ids-2017.html* ().

[59] Linux foundation. *hping3*. http://wiki.hping.org/. 2023.



Amir Al Sadi is a Ph.D. Student of Computer Science and Engineering at Alma Mater Studiorum Università degli Studi di Bologna, Bologna, Italy. He received a Master's Degree in Computer Science at the same University in 2021. His research interests focus on Software Defined Networking, Programmable Data Plane, Network Security

Marco Savi Marco Savi is a Tenure-Track Assistant Professor at the Department of Informatics, System and Communication (DISCo), University of Milano-Bicocca, Italy. In 2016 he received his Ph.D. degree in Information Technology from Politecnico di Milano, Italy and later he worked for four years at Fondazione Bruno Kessler, Trento, Italy. He joined University of Milano-Bicocca as Non-Tenured Assistant Professor in 2020 and became Tenure-Track Assistant Professor in 2022. His research interests mainly focus on the design and optimization of communication networks and on cloud computing with focus on SDN, NFV and Edge Computing. Dr Savi has been involved in various Italian/European research projects related to network softwarization, and has published around 60 papers in well-reputed journals and conferences.

Andrea Melis is an Assistant at the Department of Computer Science and Engineering at the University of Bologna. His research focuses on aspects of computer security related to innovative software architectures. In particular, his actual research activity aims to study, design, and implement innovative solutions that allow improving the safety and operational robustness of connected production industrial networks and devices for cyber-security contexts.

Marco Prandini received the Master and Ph.D. degrees in electronic and computer engineering from the University of Bologna, Italy in 1995 and 2000 respectively. He is currently an Associate Professor at the Department of Computer Science and Engineering of the same university. His research activities started in the field of public-key infrastructures and later moved to subjects related to the security of microservice-based architectures, software-defined networks, IoT, and industrial control systems.

Franco Callegati is a Full professor at the University of Bologna, Italy. His research interests are in the field of teletraffic modeling and performance evaluation of telecommunication networks. He is currently working on performance evaluation and experimental validation of SDN/NFV–based networking solutions and 5G. He has been active in EU-funded research projects since FP4. He is Senior Member of the IEEE.