Euclid: A Fully In-Network, P4-Based Approach for Real-Time DDoS Attack Detection and Mitigation

Alexandre da Silveira Ilha, Ângelo Cardoso Lapolli, Jonatas Adilson Marques, and Luciano Paschoal Gaspary, *Senior Member, IEEE*

***Abstract*—Distributed Denial-of-Service (DDoS) attacks have been steadily escalating in frequency, scale, and disruptiveness—with outbreaks reaching multiple terabits per second and compromising the availability of highly-resilient networked systems. Existing defenses require frequent interaction between forwarding and control planes, making it difficult to reach a satisfactory trade-off between accuracy (higher is better), resource usage, and defense response delay (lower is better). Recently, high-performance programmable data planes have made it possible to develop a new generation of mechanisms to analyze and manage traffic at line rate. In this article, we explore P4 language constructs and primitives to design Euclid, a fully in-network fine-grained, low-footprint, and low-delay traffic analysis mechanism for DDoS attack detection and mitigation. Euclid utilizes information-theoretic and statistical analysis to detect attacks and classify packets as either legitimate or malicious, thus enabling the enforcement of policies (e.g., discarding, inspection, or throttling) to prevent attack traffic from disrupting the operation of its victims. We experimentally evaluate our proposed mechanism using packet traces from CAIDA. The results indicate that Euclid can detect attacks with high accuracy (98.2%) and low delay (≈250 ms), and correctly identify most of the attack packets (>96%) without affecting more than 1% of the legitimate traffic. Furthermore, our approach operates under a small resource usage footprint (tens of kilobytes of static random-access memory per 1 Gbps link and a few hundred ternary content-addressable memory entries), thus enabling its deployability on high-throughput, high-volume scenarios.**

***Index Terms*—Software-defined networks, security, prototype implementation, testbed experimentation.**

I. Introduction

DISTRIBUTED Denial-of-Service (DDoS) attacks remain among the most severe threats to the security of Internet-connected systems [1]. Increases in both the frequency and the traffic volume of outbreaks have been causing significant disruptions of even large-scale online services (e.g., Imperva [2], [3], GitHub [4], and Dyn [5]). Peak data rates generated during attack campaigns amount to several terabits per second, which can saturate high-capacity links. Similarly, maximum packet rates reaching hundreds of millions of packets per second (pps) can exhaust resources on forwarding devices and attacked servers [2], [3]. There is an ongoing trend towards the worsening of the DDoS phenomenon [6]–[8], which makes it reasonable to expect even more damaging attacks in the future.

Problem Definition and Motivation: Defensive mechanisms must cater to the needs of present-day high-speed networks, whose data rates also reach the order of tens of terabits per second—especially in Internet Exchange Points (IXPs) and Service Providers (ISPs). It is a significant challenge for the mechanisms to defend these networks and their clients against DDoS attacks while meeting increasingly strict requirements for accuracy, latency, throughput, cost, and flexibility. Existing defense mechanisms seek to reach a satisfactory trade-off between these often-conflicting goals—typically, by relying on highly-specialized hardware or delegating functions to software on remote servers. Using specialized hardware, such as middleboxes based on fixed-function application-specific integrated circuits (ASICs), has the advantages of delivering high accuracy, low latency, and high throughput. However, this approach demands significant capital and operational expenditures [9], besides potentially leading to vendor lock-in and requiring forklift upgrades.

Conversely, software-based solutions are more flexible than custom-built hardware but require continuous interaction and coordination between servers and forwarding devices. Moreover, analyzing every forwarded packet in software would lead to unacceptably large overheads on processor time, memory allocation, and network management traffic. Hence, it is mandatory to diminish these overheads, which is commonly achieved by packet sampling (e.g., sFlow [10]) and flow-based accounting (e.g., NetFlow [11] and OpenFlow [12]). Despite their benefits, we advocate that these approaches still fall short in either *accuracy* or *resource usage*, depending on analysis granularity [13]. Moreover, the required coordination between data and control planes implies a long control loop, which leads to non-negligible *delays* in detection and mitigation.

Data plane programmability has emerged as a promising alternative to deal with the issues mentioned above by enabling the *in-network* execution of novel packet processing algorithms [9]. This paradigm allows a programmer to express forwarding logic as code (consisting of elementary primitives for header manipulation, memory access, and table lookup) to

Manuscript received May 18, 2020; revised October 13, 2020; accepted December 18, 2020. Date of publication December 30, 2020; date of current version September 9, 2021. This work was partially funded by the National Council for Scientific and Technological Development (CNPq - 441892/2016-7), the Coordination for the Improvement of Higher Education Personnel (CAPES - Finance Code 1), the São Paulo Research Foundation (FAPESP - 15/24494-8), the Brazilian National Research and Educational Network (RNP), and the National Science Foundation (NSF - CNS-1740911). The associate editor coordinating the review of this article and approving it for publication was K. Xue. *(Corresponding author: Alexandre da Silveira Ilha.)*

The authors are with the Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre 91501-970, Brazil (e-mail: asilha@inf.ufrgs.br; aclapolli@inf.ufrgs.br; jamarques@inf.ufrgs.br; paschoal@inf.ufrgs.br).

Digital Object Identifier 10.1109/TNSM.2020.3048265

be delegated to forwarding devices across the network. Data plane programmability makes it possible to inspect all packets of a stream directly at the data plane, thus facilitating low-latency and high-throughput network defense. Nevertheless, high-performance programmable data planes offer a restricted set of instructions, a limited amount of memory (e.g., $\approx$ tens of MB of static random-access memory—SRAM and a few MB of ternary content-addressable memory—TCAM), and a short processing time budget (tens of nanoseconds per packet) [14].

Meeting the constraints above is a difficult challenge that limits the scope of existing defense solutions. For example, Sonata [15] and Marple [16] use programmable data planes to adaptively filter packet streams intended for further examination by the control plane. In both cases, the control plane coordinates the data plane activities and executes the traffic analysis logic. As a result, these designs require continuous coordination between planes, leading to concerns about reaction time, network usage, and scalability. In turn, StateSec [17] is a data plane-based attack detection mechanism that uses OpenFlow *match+action* tables for stateful tracking of potentially harmful traffic patterns. However, its fine-grained accounting leads to high demands for memory space, limiting its deployability on high-speed switches.

Proposal: In this article, we present EUCLID, a full-fledged solution towards low-latency, fine-grained traffic inspection to detect and mitigate DDoS outbreaks. This article inherits and substantially enhances the basic ideas of a recent conference paper, building upon its learned lessons and successful results. In that paper, we introduced *DDoSD-P4* [18], an information-theoretic and probabilistic mechanism that utilizes P4 [19] to reliably *detect* the occurrence of volumetric DDoS attacks. Our mechanism relied on a statistical model based on IP address Shannon entropy to characterize legitimate traffic and calculate anomaly detection thresholds. In contrast to existing solutions that use Shannon entropy, which require a combination of general-purpose CPUs with SDN/OpenFlow switches, our mechanism is implementable in programmable forwarding devices. In addition to the Shannon-based design, this article further introduces a framework for reacting to anomaly detection alarms, integrating the *detection and mitigation* of attacks entirely into the data plane.

Two of our work's main challenges are (*i*) the adaptations and simplifications on the entropy estimation side and (*ii*) the in-depth exploration of the constructs made available by a P4 programmable data plane. As far as we are aware, our work is the first to offload this kind of anomaly detection and mitigation mechanism to programmable network devices. To meet their strict time and memory constraints, our proposed solution approximates frequencies using custom count-sketches [20] and performs compute-intensive mathematical operations with the aid of a memory-optimized longest-prefix match (LPM) table. Our method uses classifier results to apply a security policy (such as discarding, throttling, or detouring) to prevent suspicious traffic from disrupting networked services. We assess our method's efficacy through an extensive experimental evaluation, based on a proof-of-concept P4 prototype, to which we submit realistic workloads. We also compare the performance of the proposed mechanism with that of well-established solutions.

Contributions: The main contributions of this work are fourfold, as described next.

1) We push the limits of data plane programming primitives and constructs to design an *in-switch* mechanism to protect networks against DDoS attacks.
2) In contrast to existing approaches and our previous work, we design a framework that integrates both detection and mitigation of attacks in the data plane.
3) We demonstrate, employing a thorough evaluation, the performance advantages of offloading security solutions to programmable data planes.
4) We elaborate on challenges and insights that can be valuable for future research initiatives on innovative data plane-based security mechanisms.

Organization: This article is organized as follows: In Section II, we discuss related programmable data plane-based techniques to monitor networks and defend them against DDoS attacks. In Section III, we lay out the foundations for DDoS attack detection and mitigation. In Section IV, we introduce our design and discuss its implementation in a programmable switch. In Section V, we present our evaluation methodology and the results we obtained. In Section VI, we elaborate on the lessons learned during the development of this work. In Section VII, we conclude the article with final remarks and perspectives for future work.

II. Related Work

Distributed denial-of-service (DDoS) attacks and general strategies to defend networks against them have been extensively discussed in several highly-cited surveys [21]–[25]. It is a relevant challenge to defend networks against DDoS outbreaks cost-effectively, i.e., balancing requirements for performance, defense latency, and operational flexibility. One of the main challenges of defense systems is determining where to deploy their security functions (such as attack detection, attack source identification, and attack reaction [22]). At one extreme, ordinary switches would directly forward all traffic to off-path middleboxes (e.g., firewalls, intrusion prevention systems, and servers) for scrubbing (attack detection and filtering). At the other extreme, switches with advanced functionality would perform on-path traffic scrubbing by themselves without depending on middleboxes. In-between the extremes lie architectures that distribute functionalities on both off-path mechanisms and forwarding devices.

Middlebox-Based Solutions: When defense depends on middleboxes, these devices must be able to handle the high volume of traffic that flows through the switches, which leads to performance concerns. A fine-grained approach, in which all the traffic would pass through the middlebox, would be constrained by the processing and storage resources of this intermediate hop. These resource demands can be reduced by using monitoring primitives such as packet sampling (e.g., sFlow [10]) and flow-based aggregate accounting (e.g., NetFlow [11] and OpenFlow [12]). However, sampling and aggregate accounting also diminish defense

accuracy [13]. Alternatively, a middlebox based on fixed-function application-specific integrated circuits (ASICs) can achieve the desired levels of accuracy, latency, and throughput. Nevertheless, this approach demands significant expenditures as well as leads to vendor lock-in, perpetuating the so-called *network ossification* [9].

While software-based solutions running on general-purpose CPUs offer the best possible flexibility, this strategy is not future-proof. The growth in forwarding and link speeds has outpaced the increase in CPU performance, which means that scaling up requires adding extra hardware indefinitely. Numerous research efforts have sought to solve this apparent impasse by exploring the potential of software-defined networking (SDN) and programmable data planes (as we innovatively do in our work) as enablers of a new generation of security services.

SDN-Based Solutions: Several recent investigations [26]–[28] have surveyed SDN-based DDoS defense mechanisms, some of which we analyze next. Following the SDN paradigm, Xu and Liu [29] have proposed an OpenFlow-based mechanism to detect DDoS attacks and identify the source and destination hosts. In the control plane, their solution executes an unsupervised learning algorithm to classify packet flows according to their volume and rate asymmetry. The controller periodically fetches raw measurements from flow tables in the switches. Considering that the total amount of memory for these flow tables is constrained to a few thousand ternary content-addressable memory (TCAM) entries per switch, the controller dynamically adapts the data aggregation granularity to optimize memory usage while enabling zooming into abnormal traffic patterns. This adaptive process requires multiple application programming interface (API) invocations, which results in a non-negligible delay (in the order of several seconds) to detect an ongoing attack and close in on the attacking sources. Aiming to avoid the multi-second delay in detection resulting from cross-plane operations, StateSec [17] is a DDoS attack detection and mitigation mechanism that offloads all monitoring functions to the data plane. StateSec analyzes flow features (e.g., source and destination hosts and ports) through a set of in-switch finite-state machines based on extended OpenFlow tables [30]. The network controller reads the data generated by the data plane and uses an entropy anomaly-based algorithm to detect attacks. However, this approach requires a flow-table entry for every distinct flow, which potentially leads to memory saturation. The proposal does not specify how to calculate entropy within the strict time budgets (in the order of nanoseconds) of high-throughput switches. Furthermore, StateSec performs mitigation by installing, on-demand, new flow rules in the data plane to drop, queue, or deep-inspect suspect traffic. Consequently, StateSec is subject to the same delays as the method by Xu and Liu [29].

As the proposals mentioned above exemplify, SDN allows security systems to evolve. However, there is still a dependency on frequent communication between the control and data planes to carry out traffic accounting and security function processing, which results in significant overhead and delays. Security decisions are taken in the control plane time scale, requiring hundreds of milliseconds (or even whole seconds). These delays are undesirable in the context of security, for which early detection and mitigation are paramount. As opposed to these OpenFlow-based approaches, our work can perform policy enforcement in the time scale of nanoseconds.

Programmable Data Plane-Based Solutions: In contrast to the OpenFlow-based mechanisms we described above, approaches such as OpenSketch [31], UnivMon [32], and Elastic Sketch [33] fully delegate traffic accounting to the data plane. In these approaches, forwarding devices maintain summarized traffic counters in sets of hash tables, known as *sketches* [20], [34], whose values are periodically collected by the control plane. Choosing an adequate polling interval is a challenge. On one extreme, short intervals increase the network management overhead and the CPU usage in the control plane. On the other, long intervals increase the delay between the occurrence of the attack and its detection. Therefore, despite highly accurate, these solutions are subject to a trade-off between reaction time and management overhead. Nevertheless, sketches are a powerful low-footprint tool for calculating statistics on packet streams. The resource efficiency of sketches is further explored by SkyShield [35], which compares pairs of sketches related to pre- and under-attack conditions to infer the identity of malicious sources. However, SkyShield is a CPU-based security system; as such, its architecture is not suitable for high-throughput scenarios.

Aiming to offload even more monitoring logic (as compared to sampling and aggregate statistics) to the data plane, Sonata [15] provides a language for specifying packet stream filtering queries. According to operator-defined queries, programmable switches conditionally forward only the traffic of interest to external stream processors. The query language used by Sonata abstracts packet headers as tuples of field values, which can be used to define filtering and sampling rules to be executed by the data plane. Based on a similar concept, Marple [16] is a query language whose statements are compiled to target programmable forwarding devices. It also provides a new key-value store construct that enables in-network execution of functions over aggregations of packets. Marple also makes these devices able to measure traffic features; however, analyzing such metrics requires processing in external servers, which implies additional detection delay.

Other investigations have explored defenses against specific types of DDoS attacks by implementing prevention techniques on programmable networks. For instance, the most common technique involves intercepting session initiation packets in the data plane and responding to them with challenges to authenticate client hosts (via the three-way handshake) before allowing them to contact servers inside a network [36], [37]. However, this technique penalizes the connection time of all clients, even without an ongoing attack in the network. Seeking to avoid such delays and thus improve user quality-of-experience, Tavares and Ferreto [38] proposed requiring authentication only of clients trying to connect to servers that are under attack. To pinpoint these servers, the authors implemented count-sketches to estimate application-layer statistics (e.g., the number of half-opened TCP sessions) in programmable switches. Similarly, Paolucci *et al.* [39] developed a P4-based

method to detect TCP SYN flood and port-scanning attacks at edge switches. In their mechanism, upon the detection of an attack, packets identified as malicious can be dropped or steered for further inspection on an external stateful firewall. Another example of session-based defense is FrameRTP4 [28], whose anti-DDoS component uses *count-min-sketches* [40] to find heavy-hitter flows and access-control lists to block traffic related to these flows.

The main limitation of these discussed approaches is that they are general solutions primarily intended for network monitoring. While they can help defend against attacks, their applicability is limited. They operate resembling signature-based systems, observing specific field values, and detecting particular types of attacks, such as protocol exhaustion at the transport or application layers. The sketch-based approaches diminish monitoring overhead by delegating accounting to data plane devices, but they still require control plane decisions. In turn, streaming analytics, such as Sonata and Marple perform, can go beyond aggregate statistics, but security functions also depend on external stream processors. To sum up, both solution classes require frequent interaction between the control and the data plane, hindering attack detection timeliness. Unlike the approaches we analyzed above, EUCLID can be categorized as an anomaly detection system, detecting (and, very importantly, mitigating) different variations of volumetric DDoS attacks. Moreover, this process can be entirely executed in the data plane at the network line rate.

Architectures and Frameworks: We have recently seen the emergence of research on security architectures and frameworks that approach in-network defense more abstractly. For instance, Xing *et al.* [41] propose FastFlex, an architecture that implements a "multimode" data plane whose security mechanisms are enabled only when needed. Under normal conditions, switches forward data according to regular routing policies, without incurring additional latency. When under attack, switches engage their defense mechanisms to mitigate the threat. Poseidon [27] introduces a high-level language comprising a set of instructions for network monitoring and traffic management. Its users can specify defense strategies and security policies in terms of this language and deploy the resulting configurations on programmable data planes. Poseidon also includes a runtime management component that orchestrates optimal resource allocation in response to attacks whose characteristics change over time. However, despite implementing case studies for particular attack scenarios, none of these approaches focus on defense techniques. We emphasize that generic approaches, such as FastFlex and Poseidon, while good for flexibility, present similar problems to the ones that occur with CPU-based solutions. For instance, in the case of Poseidon, one needs a large amount of memory to store all the sketches demanded. In contrast, EUCLID memory requirements are minimal. Similarly to FastFlex, our approach also enables the mitigation mechanism to be activated only when needed. However, unlike that approach, EUCLID does not depend on its defense mechanisms being disabled to avoid a disproportionate impact on network latency.

Summary: The area of in-network security management is rapidly growing in importance. The flexibility of software-defined networking and programmable networks has facilitated research within much shorter design and deployment cycles. The proposals mentioned in this section represent consistent steps towards devising mechanisms to be executed in the data plane but are limited in either of two aspects. Namely, approaches that require external controllers lead to considerable communication overhead (delaying the detection of attacks) or use coarse-grained measurements to cope with the massive amount of data traversing high-speed links (degrading accuracy). On the other extreme of the design space, solutions without external controllers focus on preventing semantic attacks directed to the transport or application layers of target hosts. Since these approaches rely on state tracking, their scalability depends on the amount of memory available. A sufficiently intense attack could deplete resources in the programmable switch. In contrast, EUCLID, building upon the anomaly detection strategy proposed in our previous conference paper [18] (as stressed in the Introduction), goes a significant step further than preceding work by enabling scalable detection and mitigation of volumetric DDoS attacks completely within the data plane. Our design explores data plane programmability functionality to its full potential and achieves *accuracy*, *low intrusiveness*, and *timeliness*, as we describe in the next sections.

III. Foundations of DDoS Attack Detection and Mitigation

In this section, we revisit our previous work on DDoS attack detection and provide the groundwork for our proposed in-network DDoS defense mechanism. We begin by describing the attack scenario and the threat model we address (Section III-A). Next, we present the foundations of our attack detection strategy (Section III-B). Finally, we explain the underpinnings of the attack mitigation technique (Section III-C).

A. Attack Scenario and Threat Model

The term *distributed denial-of-service* (DDoS) refers to a broad class of attack strategies that seek to degrade the quality or disrupt the availability of services on the Internet [21]. Service degradation or disruption occurs when requests overload the target systems, either by causing network congestion or saturating computing resources. In this work, we consider a threat model whose attack vector is a large set of globally-distributed computers (e.g., a botnet) controlled by an attacker, sending illegitimate service requests to a single target host (e.g., a Web server). Moreover, the attacker uses spoofing techniques in an attempt to evade defense measures.

The attack scenario just described makes it challenging to deploy detection and mitigation mechanisms close to the attack sources—since the malicious traffic originates in several different locations, the widespread adoption of source-based defenses would require collaboration among several Internet service providers around the globe. Conversely, defenses installed on the victim's infrastructure may not be effective against the aggregated malicious traffic, which may have already saturated on-path and local network resources. Thus, we expect our

proposed mechanism to be deployed in an intermediate position within the autonomous systems (ASes) that are closest to the victim. These transit ASes typically have high-throughput forwarding devices and a privileged vantage point. Such characteristics facilitate traffic scrubbing (i.e., attack detection and mitigation) in a timely fashion before service degradation or disruption occurs. In order to prevent congestion of lower-capacity links, our mechanism should be installed on border routers, where it can analyze inter-AS traffic.

B. Traffic Characterization and Anomaly Detection

Strict performance constraints make it a significant challenge to engineer hardware for programmable switches at a reasonable cost. As a result, current programmable data planes impose strict constraints on time and memory, as well as make available only a reduced set of instructions [14]. Hence, defense mechanisms built upon this type of device must be simultaneously memory-efficient and implementable in terms of the existing programming primitives. We advocate that concepts and constructs that have already been successfully applied in the context of traffic flow analysis can help meet these goals. This is the case of entropy measurements [17], [18], [32], [33] (which we discuss in this subsection) and sketches [31]–[33], [35] (which we discuss in Section IV).

From an information-theoretic standpoint, a DDoS incident induces anomalies in the *Shannon entropy* [42] of the IP address frequency distribution. These anomalies result from increases in the total number and spread of source addresses (both legitimate and malicious) and from the concentration of traffic towards the destination address of the target host—which leads to a skewed traffic profile [25]. Thus, by accurately distinguishing between normal and abnormal traffic patterns, it is possible to detect DDoS attacks reliably [43], [44].

In this work, traffic characterization and anomaly detection begin with *frequency measurements*: our mechanism groups incoming packets in fixed-length *observation windows* (OWs), each containing *m* packets. During each OW, EUCLID counts the number of occurrences of every distinct source and destination IP address. Considering *X* the set of IP addresses within a total of *m* packets, and $f_1, f_2, \ldots, f_N$ (where $N = |X|$) the frequencies of each distinct address, the Shannon entropy of *X*, denoted by *H(X)*, is defined by

$$H(X) = \log_2(m) - \frac{1}{m}\sum_{x=1}^{N} f_x \log_2(f_x) \tag{1}$$

where the summation is the *entropy norm* of *X*, defined by

$$S(X) = \sum_{x=1}^{N} f_x \log_2(f_x)\,. \tag{2}$$

As such, Equation (1) can be rewritten as

$$H(X) = \log_2(m) - \frac{1}{m}S(X) \tag{3}$$

which highlights the negative relation between the entropy norm and the entropy itself. The minimum entropy $H(X) = 0$ occurs when all addresses are the same such that $S(X) = m\log_2(m)$. Dispersed distributions result in higher entropy values reaching the maximum $H(X) = \log_2(m)$ when all addresses are distinct, i.e., $S(X) = 0$.

EUCLID calculates entropies separately for the sets of source and destination IP addresses. During a DDoS attack, it is expected that the entropy of the set of source IP addresses increases as malicious packets introduce new values to the frequency distribution. Conversely, it is expected that the entropy of the set of destination IP addresses decreases with the victim becoming more frequent as a destination. On the one hand, these variations are only observable when the total number of packets (*m*) encompasses a sufficiently robust representation of the address frequency distributions. However, large values of *m* lead to higher attack detection delays. On the other hand, small values of *m* may render attack-related changes indistinguishable from short-term fluctuations of legitimate traffic. Our mechanism seeks to address this trade-off by scaling the entropy measurements of the source and the destination IP addresses considering a preset value of *m* (the entropy is proportional to $\log_2(m)$).

Anomaly-based attack detection has the advantage of being able to uncover attacks of unknown behavior and various intensities, although typically requiring a bootstrapping (or training) phase. In order to find relevant anomalies, our mechanism first needs to go through a training phase, during which it characterizes normal traffic. The training consists of monitoring the network for a certain number of successive *observation windows* (OWs), independently calculating source and destination IP address entropies for each OW, and summarizing these values in terms of indices of central tendency and dispersion. Our mechanism uses exponentially-weighted moving averages (EWMAs) and mean deviations (EWMMDs) [45] to summarize measurements. The source and destination entropy EWMAs are defined by:

$$M_{src,n} = \alpha H_{src,n} + (1-\alpha)M_{src,n-1} \tag{4a}$$
$$\text{with} \quad M_{src,1} = H_{src,1}\,, \text{and}$$
$$M_{dst,n} = \alpha H_{dst,n} + (1-\alpha)M_{dst,n-1} \tag{4b}$$
$$\text{with} \quad M_{dst,1} = H_{dst,1}$$

where $M_{src,n}$ and $M_{dst,n}$ are the source and destination entropy EWMAs at the observation window $n \in \mathbb{N}^*$. The factors $H_{src,n}$ and $H_{dst,n}$ are, respectively, the source and destination address entropies at OW *n*. The value $\alpha \in (0,1)$ is the *smoothing coefficient*—a parameter that allows us to filter short-term fluctuations while giving prominence to the most recent entropy measurements. The averages are initialized with the first entropy measurements.

The source and destination EWMMDs are defined by:

$$D_{src,n} = \alpha|M_{src,n} - H_{src,n}| + (1-\alpha)D_{src,n-1} \tag{5a}$$
$$\text{with} \quad D_{src,1} = 0\,, \text{and}$$
$$D_{dst,n} = \alpha|M_{dst,n} - H_{dst,n}| + (1-\alpha)D_{dst,n-1} \tag{5b}$$
$$\text{with} \quad D_{dst,1} = 0$$

where $D_{src,n}$ and $D_{dst,n}$ are the source and destination EWMMDs at the OW *n*. The factors $M_{src,n}$ and $M_{dst,n}$,

respectively, are the source and destination EWMAs at OW *n*. The value $\alpha \in (0, 1)$ is the smoothing coefficient. The mean deviations are initialized with zero.

After having obtained a model of normal traffic under safe conditions, we use its EWMAs and the EWMMDs to decide whether an entropy measurement is anomalous, in which case our mechanism triggers a DDoS attack alarm; otherwise, we update the traffic model. An entropy anomaly occurs when any of the following inequalities hold:

$$H_{src,n} > M_{src,n-1} + kD_{src,n-1} \tag{6a}$$

$$H_{dst,n} < M_{dst,n-1} - kD_{dst,n-1}\,. \tag{6b}$$

The value *k* is a configurable parameter called the *sensitivity coefficient*, which scales the detection threshold. Since *k* multiplies the index of dispersion, its effect is directly proportional to the variability in traffic patterns. Lower values of *k* allow the detection of subtler attacks, at the cost of a lower *specificity* (i.e., a higher number of legitimate fluctuations incorrectly interpreted as attacks). Conversely, higher values of *k* result in higher statistical specificity at the cost of letting less-relevant attacks remain unnoticed. It is the responsibility of the network administrators to choose *k* values according to the intended level of accuracy (i.e., high sensitivity and high specificity). The experimental results from our previous work [18] indicate that when *k* is in the ideal operating range, our mechanism can accurately (i.e., $\approx 98\%$ sensitivity and $\approx 90\%$ specificity; see Section V-C) detect and signal the occurrence of DDoS attacks. Such a high accuracy allows us to use our detection alarms as trustworthy inputs for a mitigation mechanism—which we discuss in the following subsection.

C. Inferring Intent From Frequency Variation Anomalies

We use the output of our anomaly detection component as a trigger to (*i*) identify which packet sources are the likely culprits of an attack and (*ii*) apply a suitable countermeasure. We design our mechanism following the observation that Shannon entropy anomalies are likely caused by addresses whose frequencies have excessively diverged from baseline measurements [22].

The difference between the frequency variations of legitimate and malicious addresses is significant enough to allow the accurate classification of packets. By finding an adequate threshold, we can obtain acceptable results for *sensitivity* (i.e., the true-positive rate or proportion of correctly-identified malicious packets) and *specificity* (i.e., the true-negative rate or proportion of correctly-identified legitimate packets).

Since we are interested in comparing safe and unsafe network conditions, we need to keep track of these. We consider a network *safe* when none of its nodes is undergoing a detectable DDoS attack; otherwise, we consider the network *unsafe*. When the anomaly detection module indicates that the network is unsafe, the mechanism enters a *defense readiness* mode, in which it will remain until the network has been safe for a predetermined *cooldown interval*—which avoids premature defense de-escalation.

We model defense readiness as a finite state machine (FSM) that controls the operation of the attack mitigation mechanisms. At the end of each observation window, the mechanism updates the FSM state according to the safety of the network in W_{last} (the OW whose accounting has just finished). Figure 1 illustrates the defense-readiness FSM. It starts in the `SAFE` state, in which detection is active, but mitigation is dormant. Whenever an attack is detected in W_{last}, the FSM transitions to the `DEFENSE ACTIVE`, in which mitigation is active. The FSM remains in the `DEFENSE ACTIVE` until no attack is detected in *Wlast*, in which case there is a transition to `DEFENSE COOLDOWN`. Once in cooldown, mitigation remains active; however, if no attack has been detected for a predetermined number of OWs (not shown in the figure), the machine transitions to `SAFE`.

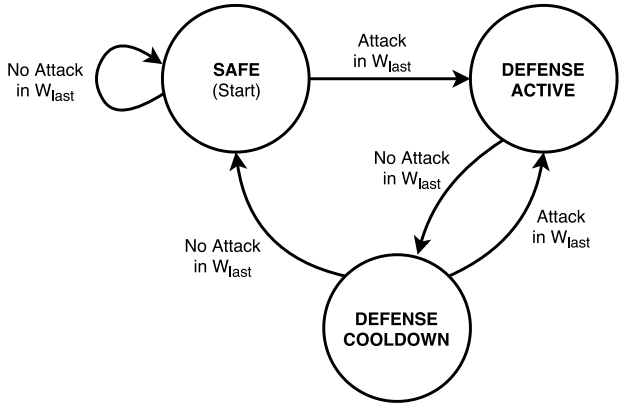


Fig. 1. Defense-Readiness State Machine.

Once in any of the `DEFENSE` states, the defense pipeline calculates the *frequency variation* for each incoming packet. The frequency variation is denoted by *V* and defined by

$$V = V_{dst} - V_{src} \tag{7}$$

where V_{src} and V_{dst} measure the changes in frequencies of the source (*src*) and destination (*dst*) addresses of the packet. During an attack, we expect relevant changes in V_{src} and V_{dst}. The intuition behind this is that variations related to legitimate traffic will be proportional for both source and destination addresses. However, for the malicious traffic, this pattern changes: while the total traffic grows, the frequencies of the legitimate source addresses will vary disproportionately to the frequencies of the malicious addresses. Relative frequencies for source hosts tend to decrease (many hosts). Relative frequencies for attack targets tend to increase. By subtracting V_{src} from V_{dst}, we expect to obtain larger values of *V* for malicious packets than for legitimate ones. We calculate the variations between the last observation window (W_{last}) and the observation window used as a baseline of a safe network condition (W_{safe}). The values of the *V*-terms are given by

$$V_{src} = f_{src,last} - f_{src,safe} \text{ and} \tag{8a}$$

$$V_{dst} = f_{dst,last} - f_{dst,safe} \tag{8b}$$

where $f_{src,last}$ and $f_{src,safe}$ are the frequencies of *src* in W_{last} and W_{safe}, respectively. Similarly, $f_{dst,last}$ and $f_{dst,safe}$ refer to the destination address.

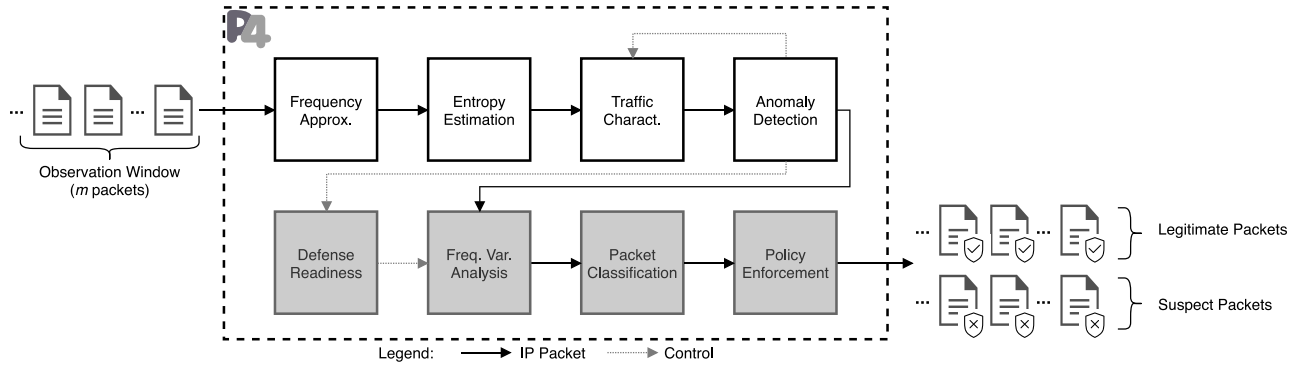


Fig. 2. Anti-DDoS Attack Mechanism Top-Level Scheme.

After calculating the frequency variation, classification occurs according to a mitigation threshold t and annotates (by setting metadata) each packet as *legitimate* or *suspect*:

$$V \leq t \Rightarrow \text{packet is legitimate;} \tag{9a}$$

$$V > t \Rightarrow \text{packet is suspect.} \tag{9b}$$

Once packets have been annotated, the mechanism is ready to enforce an operator-configurable security policy, such as *discarding*, *throttling*, or *detouring*. Discarding is straightforward: we immediately drop the suspect packet. Throttling consists of forwarding suspect packets to a different egress queue, with a limited number of entries and a predefined dispatch rate, thus limiting the volume of suspect traffic allowed to reach their target. Detouring enables various scenarios: packets forwarded to a different path can undergo, for instance, stateful, light, or deep inspection.

In the next section, we present the design and implementation of the packet processing pipeline that materializes the defense strategy we have described.

IV. Our Design for In-Network DDoS Attack Detection and Mitigation

In this section, we discuss the design and implementation of the EUCLID packet processing pipeline, which conforms to the foundations presented in Section III.

We implement EUCLID in $\mathrm{P4}_{16}$ [19]. By using this language to specify our mechanism, we facilitate its deployment on compatible P4-programmable switches suitable for high-speed, high-throughput packet forwarding. Notwithstanding the versatility of the language, P4 programs must operate under strict constraints, such as a reduced set of instructions and a limited amount of memory—in the order of megabytes of static random-access memory (SRAM) and kilobytes of ternary content-addressable memory (TCAM). Consequently, to perform real-time anti-DDoS defense, we need to satisfy these architectural constraints. Hence, we designed EUCLID with resource-efficiency in mind: its operation requires, for each 1-Gbps link, less than 80 KB of SRAM and 2 KB of TCAM. We discuss the requirements for scaling up our mechanism to meet the needs of faster links in Section V-C2.

We present an overview of EUCLID in Figure 2. The top row portrays the attack detection components proposed in the context of our previous work [18]. Throughout its operation, our mechanism partitions the stream of incoming packets into fixed-size *observation windows* (OWs). During the processing of each OW, the *frequency approximation* component tallies the frequency (number of occurrences) of every source and destination IP address (Section IV-A1). Next, these frequencies are used as inputs by the *entropy estimation* logic (Section IV-A2). At the end of the OW, the *traffic characterization* component uses the entropy estimates to build and update a statistical model of normal traffic conditions (Section IV-A3). Then, the *anomaly detection* stage employs the traffic model to check for abnormal changes in IP address entropies, in which case it issues attack alarms (Section IV-A4).

Also, in Figure 2, the bottom row lays out the *attack mitigation* components we presently introduce. Attack alarms trigger transitions in a *defense-readiness* state machine, which activates and coordinates the operation of the remaining components (Section IV-B1). When attack mitigation is enabled, every packet undergoes three stages. First, EUCLID analyzes the address accounting history to measure the *frequency variation* of the IP addresses (Section IV-B2). Next, the *packet classification* section decides whether there have been unwarranted changes in frequency, in which case it labels packets as suspects (Section IV-B3). Last, *policy enforcement* applies network-operator-defined rules to determine the adequate destination for the packet (Section IV-B4). In the remainder of this section, we detail the implementation of each attack detection and mitigation component.

A. Collecting Traffic Statistics

Our anomaly-based attack detection strategy depends on traffic characterization, which requires measuring the Shannon entropies of the sets of source and destination IP addresses in each OW (Section III-B). The entropies, in turn, depend on statistics about the frequency of every source and destination IP address. Keeping this kind of accounting is potentially computing- and storage-intensive, especially when exact measurements are required. To make it feasible to obtain these quantities under the processing constraints of a programmable data plane, EUCLID builds these statistics through the *frequency approximation* and *entropy estimation* pipeline, which we detail next. Figure 3 illustrates the entire pipeline.

1) Frequency Approximation: We approximate address frequencies by employing count-sketches [20], which we

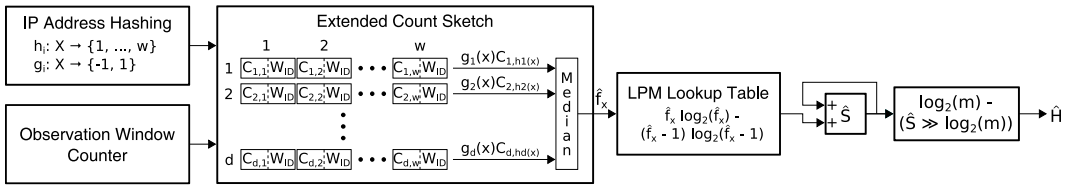


Fig. 3. Entropy Estimation Pipeline.

briefly discussed in Section II. Count-sketches are data structures that provide a compressed representation of frequency tables of events in data streams. In our design, the arrival of a packet in the switch corresponds to two separate events: one for the source and one for the destination IP address. Moreover, the data stream consists of all the events within a given OW. The total size needed for a count-sketch to represent events without compression grows sublinearly in *m* [40]. Nonetheless, sketches provide unbiased frequency estimates within parameterizable probability and tolerances.

Formally, a count-sketch is an abstract data type represented by a tuple (C, X, F_h, F_g) and two operations, UPDATE and ESTIMATE, defined as follows. Let *X* be the set of all possible IP addresses and *C* be a two-dimensional matrix of counters with depth d and width w (i.e., $C \in \mathbb{Z}^{d\times w}$), where $C_{i,j}$ denotes the counter at row i and column j (see Figure 3). We define two sets of independent hash functions $F_h = \{h_1, \ldots, h_d\}$ and $F_g = \{g_1, \ldots, g_d\}$, where each ordered pair $(h_i, g_i) \in F_h \times F_g$ is associated with a matrix row $i \in \{1, \ldots, d\}$. All hash functions take an IP address $x \in X$ as a parameter. Hash function h_i maps addresses to column numbers in row i (i.e., $h_i{:}X \mapsto \{1, \ldots, w\}$). Hash function g_i determines whether the counter $C_{i,h_i(x)}$ should be incremented or decremented (i.e., $g_i{:}X \mapsto \{-1, 1\}$).

The count-sketch operations are summarized as follows:

$$\begin{aligned}
&\textsc{Update}(C, x):\\
&\quad \text{for } i = 1, \ldots, d:\\
&\quad\quad C_{i,h_i(x)} \leftarrow C_{i,h_i(x)} + g_i(x)\\
&\textsc{Estimate}(C, x):\\
&\quad \text{return } \ median\Big(g_1(x)C_{1,h_1(x)}, \ldots, g_d(x)C_{d,h_d(x)}\Big)
\end{aligned}$$

UPDATE*(C, x)* counts an occurrence of *x* by updating exactly one entry in each of the *d* depth levels of the sketch *C*. ESTIMATE*(C, x)* returns an estimate of the frequency count of *x*, which we denote as $\hat{f}_x$ (see Fig. 3).

The count-sketch uses the hash functions g_i to treat h_i collisions for multiple distinct IP addresses. Expectedly, when collisions occur, some addresses will increase the counter, and others will decrease it, which would result in inconsistent estimates. However, when considering all *d* counters for a given IP address, counters whose values are affected by collisions become outliers. By calculating the median (which eliminates outliers) of the values stored in all rows, the count-sketch avoids generating biased frequency estimates. It is essential to notice that we need to implement a median operator whose number of inputs is equal to the sketch depth. Consequently, *d* directly influences the complexity of the median calculation, which requires $\mathcal{O}(d^2)$ execution steps in order to compare all inputs. In P4, we specify the count-sketch matrices as registers, which allow the stateful storage of general-purpose data. We implement the IP address hashing operations as custom hash functions. In our design, hash functions are homomorphic to $h(x) = (a_i x + b_i) \bmod p$, where a_i and b_i are co-prime coefficients, and *p* is a prime number. This class of functions is suitable for deployment in programmable data planes, as previous work demonstrates [46].

Our design requires calculating independent frequency approximations for each OW, which mandates resetting all count-sketches before the first usage within a window. To avoid bursty processing overheads, we resort to *extended count-sketches*. In our implementation, we associate an additional register to each sketch entry (see Figure 3), where we store the index of the OW in which it was last updated (W_{ID}). This index, in turn, comes from a P4 stateful counter. Thus, whenever our mechanism reads an extended count-sketch, outdated entries are presumed zero and updated accordingly.

Once EUCLID has processed an incoming packet and updated its frequency approximation, it can proceed to the entropy estimation step.

2) Entropy Estimation: Considering P4 has no support for floating-point arithmetic, EUCLID stores and handles measurements in a fixed-point format, which allows obtaining fractional precision using only integer operations. Given that P4 also lacks instructions to calculate binary logarithms, we simplify the calculation of Equation (1). For the first term, we set the observation window size *m* to a fixed (parameterizable) value so that $\log_2(m)$ becomes a constant. As a result, the real-time entropy estimation processing requires only the calculation of the second term, i.e., the entropy norm, as we explain next.

a) Entropy norm estimation: The second term of Equation (1) is given by $S(X) = \sum_{x=1}^{N} f_x \log_2(f_x)$. This term is a function of the frequencies of each distinct IP address observed in the window (recall that the calculations are independently done for source and destination IPs). After the pipeline reads an IP address and updates its approximate frequency $\hat{f}_x$, it is ready to compute the corresponding term in the entropy norm estimate $\hat{S}$ (to simplify notation, we omit the parameter *X*). As each IP address is expected to occur numerous times in each OW, the pipeline updates $\hat{S}$ by adding to it the difference between the newly-computed term and its previous value. This is done for $\hat{f}_x > 1$, as follows:

$$\hat{S} \leftarrow \hat{S} + \underbrace{\hat{f}_x \log_2\Big(\hat{f}_x\Big)}_{\text{newly-computed term}} - \underbrace{\Big(\hat{f}_x - 1\Big)\log_2\Big(\hat{f}_x - 1\Big)}_{\text{previous term value}}. \tag{10}$$

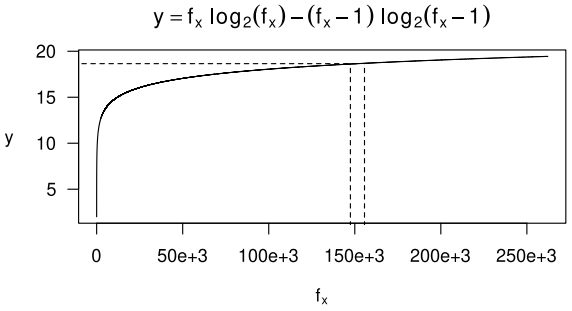


Fig. 4. LPM lookup table pre-computed function: the dashed lines illustrate how f_x values can be aggregated to a single table entry with reduced approximation error.

To calculate Equation (10), we define a pre-computed function (PCF), which we implement as a longest prefix match (LPM) lookup table. Our LPM table contains values for $\hat{f}_x \log_2(\hat{f}_x) - (\hat{f}_x - 1)\log_2(\hat{f}_x - 1)$. Unlike an exact lookup table, which would require an entry for each domain value, the LPM maps variable-length intervals of domain values to a single entry. As an LPM is typically implemented in a switch by a ternary content-addressable memory (TCAM), we eliminate the need for real-time multi-step operations, replacing them with a single-step TCAM lookup.

We plot our pre-computed function in Figure 4. The dashed lines represent the aggregation of the domain interval [147 456, 155 647] to a single entry whose image is $y = 18.65214$. In this case, the maximum approximation error is ≈ 0.04 when $f_x = 147\,456$. In general, the magnitude of the error is directly proportional to $^{dy}/_{df_x}$, i.e., $\log_2(f_x) - \log_2(f_x - 1)$, whose value decreases as f_x grows. Consequently, aggregation intervals can be larger for higher frequencies. In our reference implementation,[1] we propose an algorithm to populate the LPM while meeting an adequate trade-off between entry count and maximum error.

Throughout the operation of our P4-based design, every incoming packet *x* triggers UPDATE*(C, x)* and $\hat{f}_x \leftarrow$ ESTIMATE(C, x) (for source and destination IPs). Our mechanism uses $\hat{f}_x$ as a key to look up the increments to the entropy norms. When our mechanism reaches the end of each observation window, it uses $\hat{S}$ to estimate the entropy $\hat{H}$ (for conciseness, we omit the parameter *X*), as shown below.

b) Entropy measurement: Seeking to further diminish the processing requirements, we constrain the operation window size *m* to a fixed power of two. Thus, $\log_2(m)$ yields an integer constant, which makes it possible to calculate $\hat{S}/m$ as an arithmetic shift. The resulting expression for the entropy estimate is:

$$\hat{H} \leftarrow \log_2 m - \left(\hat{S} \gg \log_2 m\right) \tag{11}$$

where $\gg$ denotes an arithmetic shift. We store the value $\log_2(m)$ in a register to allow the parameterization of *m* at runtime.

In a recently-published work, Ding *et al.* [47] have introduced algorithms to calculate logarithmic functions and entropy estimates in programmable data planes. Differently from our work, their approach does not require a TCAM-backed pre-computed function (PCF). The solution requires additional processing steps for each packet—which potentially implies the allocation of more pipeline stages. The analysis of the suitability of this solution as a substitute for our PCF-based approach (regarding the processor time and memory space trade-off) exceeds the scope of our current work.

After the entropy estimation phase finishes, the switch begins processing the next functional components—traffic characterization and anomaly detection.

3) Traffic Characterization: We summarize entropy measurements in terms of their EWMAs and EWMMDs (Section III-B, Equations (4a), (4b), (5a), and (5b)). As in the case of entropy estimation, we use fixed-point notation. We choose different representations to allow for sufficient numeric precision. For instance, whereas we represent entropy measurements as 28 integer and 4 fractional bits, we store moving averages and deviations as 14 integer and 18 fractional bits. For the smoothing coefficient α, eight fractional bits are sufficient. In order to preserve precision, we take special care to specify operation order and binary radix point alignment.

4) Attack Detection: As in the traffic characterization component, attack detection uses fixed-point arithmetic to calculate source and destination thresholds. The sensitivity coefficient *k* is represented with five integer and three fractional bits. We check whether the last entropy measurements exceed these thresholds according to Equations (6a) and (6b). If, and only if, both entropy estimates fall within the dynamically-calculated thresholds, we update the traffic model. Conversely, if an anomaly is detected, the switch records such occurrence by setting a packet metadata field. This enables the generation of a signaling packet and triggers a state transition in the defense-readiness FSM, which we discuss next.

B. Attack Mitigation

The attack mitigation mechanism, formalized together with its detection counterpart in Algorithm 1, follows the principles we discussed in Section III-C. The anomaly detection component triggers state transitions in the defense-readiness finite-state machine (FSM). The FSM, in turn, directs the operation of all the attack mitigation logic. Next, we discuss the implementation of the state machine and the remaining security stages.

1) Defense Readiness: In this stage, executed once for each observation window (OW), EUCLID checks the attack alarm and the defense-readiness (DR) state to perform a conditional transition (Algorithm 1, Lines 29-34). If the attack alarm metadata field is set, the state machine transitions to `DEFENSE ACTIVE`, as explained in Section III-C. In contrast, if the attack alarm flag is inactive, there are two possibilities: (*i*) when in `DEFENSE ACTIVE`, DR moves to `DEFENSE COOLDOWN`, in which it remains for an additional predetermined number of observation windows (in our implementation, we set this number to one); (*ii*) when already in `DEFENSE COOLDOWN`, DR transitions back to `SAFE`. After DR executes, its resulting state lasts at least until the end of the next OW, when new checks and possibly new transitions

[1] Available at https://www.github.com/aclapolli/ddosd-cpp.

Algorithm 1 Attack Detection and Mitigation

Input: P ▷ Packet headers and metadata

A.1) Frequency Approximation.

1: **for** $i \in [1, 2, ..., d]$ **do**
2: $h_{src}(i) \leftarrow h_i(P.src)$
3: $g_{src}(i) \leftarrow g_i(P.src)$
4: **if** $C_{src}(i, h_{src}(i)).W_{ID} \neq W_n$ **then**
5: **if** $W_n > 1$ **and** $DR_{state} = \texttt{SAFE}$ **then**
6: $C_{src,safe}(i, h_{src}(i)) \leftarrow C_{src,last}(i, h_{src}(i))$
7: $C_{src,last}(i, h_{src}(i)) \leftarrow C_{src}(i, h_{src}(i))$
8: $C_{src}(i, h_{src}(i)) \leftarrow 0$
9: $C_{src}(i, h_{src}(i)).W_{ID} \leftarrow W_n$
10: $C_{src}(i, h_{src}(i)) += g_{src}(i)$
11: $f_{src} \leftarrow median(\{g_{src}(i)C_{src}(i, h_{src}(i))|\forall i \in [1, 2, ..., d]\})$
12: $S_{src} \leftarrow S_{src} + \text{PCF}(f_{src})$

The same procedure presented in Lines 1-12 is similarly carried out for the destination address. Omitted for space.

13: $PC \leftarrow PC + 1$
14: **if** $PC = (2^{\log_2(m)})$ **then**
15: $W_n \leftarrow W_n + 1$

A.2) Entropy Estimation.

16: $H_{src} \leftarrow \log_2(m) - (S_{src} >> \log_2(m))$
17: $H_{dst} \leftarrow \log_2(m) - (S_{dst} >> \log_2(m))$

A.3) Traffic Characterization and
A.4) Anomaly Detection.

18: **if** $W_n = 1$ **then**
19: $M_{src} \leftarrow H_{src}$; $M_{dst} \leftarrow H_{dst}$
20: $D_{src} \leftarrow 1$; $D_{dst} \leftarrow 1$
21: **else**
22: $A \leftarrow (H_{src} > (M_{src} + kD_{src})$ **or** $H_{dst} < (M_{dst} - kD_{dst}))$
23: **if** A **is** *False* **then**
24: $M_{src} \leftarrow \alpha H_{src} + (1 - \alpha)M_{src}$
25: $M_{dst} \leftarrow \alpha H_{dst} + (1 - \alpha)M_{dst}$
26: $D_{src} \leftarrow \alpha|H_{src} - M_{src}| + (1 - \alpha)D_{src}$
27: $D_{dst} \leftarrow \alpha|H_{dst} - M_{dst}| + (1 - \alpha)D_{dst}$
28: $P_n \leftarrow 0$; $S_{src} \leftarrow 0$; $S_{dst} \leftarrow 0$

B.1) Defense-Readiness.

29: **if** A **is** *True* **then**
30: $DR_{state} \leftarrow \texttt{ACTIVE}$
31: **else if** $DR_{state} = \texttt{ACTIVE}$ **then**
32: $DR_{state} \leftarrow \texttt{COOLDOWN}$
33: **else if** $DR_{state} = \texttt{COOLDOWN}$ **then**
34: $DR_{state} \leftarrow \texttt{SAFE}$

B.2) Frequency Variation Analysis and
B.3) Packet Classification.

35: $P.metadata.classification \leftarrow \texttt{LEGITIMATE}$
36: **if** $DR_{state} \neq \texttt{SAFE}$ **then**
37: $f_{src,last} \leftarrow \text{ESTIMATE}(C_{src,last}, P.src)$
38: $f_{src,safe} \leftarrow \text{ESTIMATE}(C_{src,safe}, P.src)$
39: $f_{dst,last} \leftarrow \text{ESTIMATE}(C_{dst,last}, P.dst)$
40: $f_{dst,safe} \leftarrow \text{ESTIMATE}(C_{dst,safe}, P.dst)$
41: $V_{src} \leftarrow f_{src,last} - f_{src,safe}$
42: $V_{dst} \leftarrow f_{dst,last} - f_{dst,safe}$
43: $V \leftarrow V_{dst} - V_{src}$
44: **if** $V > t$ **then**
45: $P.metadata.classification \leftarrow \texttt{MALICIOUS}$

B.4) Policy Enforcement.

46: **if** $P.metadata.classification$ **is** LEGITIMATE **then**
Apply the normal forwarding table.
47: **else**
Apply the mitigation forwarding table.

Output: P

will occur. In both DEFENSE states, EUCLID submits every incoming packet to the subsequent stages—frequency variation analysis, packet classification, and policy enforcement, which we discuss next.

2) Frequency Variation Analysis: In this component, we follow the observation that entropy anomalies are more likely due to excessive occurrences of the IP addresses of the attackers, whose frequencies have varied the most between a reference, baseline OW, and the OW in which we detected an attack (Section IV-B2). By uncovering these highly-divergent addresses, we can identify the attack sources. Hence, we change the source identification problem to a matter of finding excessive variations. EUCLID already performs frequency approximation to calculate entropies. We extend that principle to accurately pinpoint sources of malicious traffic.

Similarly to the frequency approximation stage, this step also uses count-sketches to obtain approximate quantities. However, in this component, we process historical data, i.e., counts obtained in W_{last} and W_{safe} (respectively, the previous observation window and the last OW during which the network was safe, as defined in Section III-C). We store this data in four additional count-sketches: two for source addresses, two for destination addresses, i.e., $C_{src,last}$, $C_{dst,last}$, $C_{src,safe}$, and $C_{dst,safe}$. We further extend the count-sketch by adding to it the COPY operation:

$$\begin{aligned}&\text{COPY}(CTarget, COrigin, x):\\ &\quad\text{for } i = 1, \ldots, d:\\ &\qquad CTarget_{i,h_i(x)} \leftarrow COrigin_{i,h_i(x)}\end{aligned}$$

COPY*(CTarget, COrigin, x)* iterates on all depth levels of the target and origin sketches to copy the counters associated with a given IP address *x* (Algorithm 1, Lines 6-7).

EUCLID uses count-sketches for this purpose instead of alternatives such as count-min-sketches (CMS), which would suffice for frequency variation analysis and could be faster [48]. The reason for our choice is twofold. First, EUCLID's entropy estimation component requires unbiased frequency estimates for accurate detection, which the CMS does not provide. Second, at this point in execution, EUCLID has already calculated the values of the sixteen hash functions needed for the main count-sketches (which we store in arrays as exemplified in Lines 2-3) and can simply re-use these values to copy data to the historical count-sketches. Using another type of sketch would require computing sixteen additional hash functions for each packet, increasing the computing resources footprint.

Right before executing the frequency approximation steps, we follow a procedure to ensure the updating of the counters related (*i*) to the current OW (stored in C_{src} and C_{dst}); (*ii*) to the last OW (stored in $C_{src,last}$ and $C_{dst,last}$); (*iii*) and to the baseline OW (stored in $C_{src,safe}$ and $C_{dst,safe}$). Since the counters from W_{last} and W_{safe} do not change more than once per OW, we only perform the corresponding operations at the *first* occurrence of x_{src} (resp., x_{dst}) in W_{curr}. Moreover, to avoid having to allocate memory for temporary data, we perform the operations in the order specified in Lines 5-7 of Algorithm 1. Furthermore, we only update the counters from W_{safe} if the DR state is SAFE (Line 5). Back at the frequency variation analysis component, we perform the calculations based in Equations (7), (8a), and (8b) (Section III-C), as indicated in Lines 37-43. At this point, EUCLID is ready to proceed to the packet classification component.

3) Packet Classification: For every packet that goes through the data plane when the defense state is ACTIVE or

COOLDOWN, EUCLID calculates the frequency variation $\hat{V}$. As discussed in Section III-C, we expect that legitimate packets have smaller $\hat{V}$ values than the malicious packets have. By applying a mitigation threshold (t), our mechanism attempts to identify packets as legitimate or malicious. We seek optimal results both for the true-positive rate (TPR, the proportion of malicious packets correctly identified) and the false-positive rate (FPR, the proportion of legitimate packets mistaken as malicious). An ideal TPR is close to 100% so that our mechanism can correctly submit most malicious packets to the countermeasures they require. In contrast, the FPR must be close to zero percent, so that legitimate traffic does not suffer undue disruption.

The mitigation threshold t is parameterizable by the network operator. We envision the dynamic adjustment of this threshold as future work. Once defined, t is used in a test: if $\hat{V} > t$, our mechanism sets a metadata field to flag the packet as suspect (Section III-C, Equation (9b)) (Lines 44-45). Thus, it sets up the packet to be processed by the next component—policy enforcement. Otherwise, i.e., if $\hat{V} \leq t$, the switch proceeds to perform its ordinary forwarding functions.

4) Policy Enforcement: The last stage of our security pipeline is policy enforcement (Lines 46-47). At this point, we apply a match-action table to determine how further processing of the packet must occur. Our design allows the network operator to choose between three policies to be applied to suspect packets: `discard`, `throttle`, and `divert`. The `discard` policy is implemented by directly calling the P4 `drop` primitive. The `throttle` policy sends the packet to a rate-limited egress queue (although more elaborate implementations are viable). The `divert` policy changes the egress interface so that the packet can be processed off the main path by different devices (e.g., a deep packet inspector). The network operator selects policies by populating a match-action table with applicable rules.

V. EVALUATION

To the best of our knowledge, EUCLID is the first work to explore data plane programmability, more specifically P4, to devise a sophisticated DDoS attack detection and mitigation mechanism. Due to the constraints related to the reduced set of $\mathrm{P4}_{16}$ programming primitives, implementing our design requires numeric approximations and compact data representations (i.e., sketches). Hence, it is imperative to assess the *accuracy*, *resource utilization*, and *responsiveness* of our proposed mechanism thoroughly. In this section, we seek answers to the following research questions (RQs):

- *RQ1: How accurate is the entropy estimation pipeline as a function of memory space requirements?*
- *RQ2: Assuming reliable entropy estimates (RQ1), how accurate is our detection mechanism under different settings and attack intensities?*
- *RQ3: How accurate and responsive is our detection mechanism as compared to other monitoring strategies?*
- *RQ4: Assuming reliable detection (RQ2), how effective is our attack mitigation mechanism under different settings?*

Collectively, these questions prompt us to investigate to what extent and under which conditions it is possible to rely on a fully in-network approach to obtain protection against DDoS attacks.

In the next subsection, we detail our evaluation methodology and experimental setup. Right after, we discuss the results that support us in answering the research questions above. Finally, we elaborate on how our findings relate to our mechanism's applicability to various attack scenarios.

A. Evaluation Methodology and Experimental Setup

This section describes the topology, testbed, and traffic generation methods and also details our experimental design.

Topology and Target Devices: Recall from Section III-A that we expect our proposed mechanism to be deployed in an intermediate position within the autonomous systems (ASes) that are closest to the victim. Also, to prevent congestion of lower-capacity links, our mechanism should be installed on border routers (in each traffic ingress point), where it can analyze inter-AS traffic. Given these scenarios, without loss of generality, we use a single forwarding device (assuming traffic enters the network through one point only, which is the case of numerous setups). The single switch represents the point at which we deploy EUCLID.

We designed our solution for deployment on an RMT-based [14] hardware device (e.g., a Barefoot Tofino switch [49]). However, due to the relatively recent introduction of the P4 language and the small number of P4-programmable switch suppliers, this hardware has yet to become an off-the-shelf commodity. Despite this, software solutions facilitate the progress of the research on programmable networks. In this work, we conduct our experiments on a software-based $\mathrm{P4}_{16}$ testbed [50]. This setup does not affect our evaluation since both the accuracy and resource utilization are target-independent (i.e., hardware and software targets are functionally equivalent). Furthermore, by design, a hardware RMT-based pipeline forwards packets at line-rate and within a fixed delay between ingress and egress (if there is no recirculation) [51]. Since EUCLID does not use recirculation, timing is, therefore, not of concern.

The EUCLID source code is available at our Github repository[2] and can be used as a starting point for new developments in the area. Our repository also includes the data analysis notebooks, scripts, and tools we used for this work.

Datasets and Traffic Generation Strategy: We perform a packet trace-driven evaluation using representative datasets of legitimate and malicious traffic. As legitimate traffic, we use the *CAIDA Anonymized Internet Traces 2016* [52] dataset, recorded from high-speed Internet backbone links. As attack traffic, we use the *CAIDA DDoS Attack 2007* [53] dataset, which consists of an attempt to deplete the computing resources of a target server and to saturate its connection to the Internet. Despite not recent, the DDoS dataset is renowned for its thoroughness and applicability to assess system performance under attack, as several high-impact publications on network security have attested. The volumetric

[2]Available at https://www.github.com/asilha/euclid.

TABLE I
SYSTEM FACTOR LEVELS

	Levels Used in Each Subsection			
System Factors	**V-B**	**V-C**	**V-D**	**V-E**
Observation Window Size (m)	2^{18}	2^{18}	2^{18}	$\{2^{14}; 2^{16}; 2^{18}\}$
Hashing Coefficients (a_i, b_i)	pseudo-random and pairwise-independent			fixed
Count-Sketch Depth (d)	$\{4, 8, 16\}$	4	4	4
Count-Sketch Width (w)	$\{64, 368, 672, 976, 1280\}$		1280	1280
Sensitivity Coefficient (k)	NA	$\{0, 0.5, 1, ..., 8\}$	4	$\{4.875, 4.875, 3.625\}$
Defense Threshold (t)	NA	NA	NA	$64k, k \in \{\mathbb{Z} \mid -16 \leq k \leq 16\}$

DDoS attack captured in the dataset matches the attack scenario described in Section III-A.

We use our traffic generator TRAFG[3] to combine the aforementioned datasets, forming synthetic workloads. Each workload follows a common structure: a *training phase* and a *detection and mitigation phase*. We set the length of detection and mitigation phase n to 2^{27} packets and that of the training phase to *n/2* packets. During the training phase, EUCLID analyzes only legitimate traffic in order to initialize the characterization model. The detection and mitigation phase is subdivided into three segments: *pre-attack*, with *n/4* legitimate packets, *attack*, with *n/2* packets (both legitimate and malicious), and *post-attack*, with *n/4* legitimate packets. The attack segment combines *pn/2* malicious packets and $(1-p)n/2$ legitimate packets, randomly selected according to a Bernoulli distribution with probability p. By varying p (e.g., 3%, 3.5%,..., 6%, 20%), we represent different attack intensities (i.e., the proportion of malicious packets to the total number of packets during the attack). The TRAFG generator takes as inputs the datasets and the parameters n and p, and it outputs a packet trace file that follows the structure we described. Once we have the workloads, each experiment consists of submitting the packets into a switch interface.

Experimental Design: Table I shows the system factor levels we use throughout the evaluation. To evaluate estimation accuracy and detection performance (RQ1-RQ3), we set the observation window length m to 2^{18} packets, which corresponds to approximately 250 ms of traffic at 1Mpps (the mean packet rate of our workload). To evaluate mitigation performance (RQ4), we use three different observation window sizes ($m \in \{2^{14}, 2^{16}, 2^{18}\}$ packets). These sizes allow us to investigate the effect of the window size on the detection and mitigation delays, as well as to memory usage. While 2^{18} packets represent $\approx$ 250 ms of traffic, 2^{16} packets take $\approx$ 65 ms, and 2^{14} packets take $\approx$ 15 ms. We define varying value ranges for count-sketch dimensions (d and w), sensitivity coefficient (k), and observation window size (m). These variations enable a broad assessment of EUCLID under different configurations. As detection relies on hash functions (for the address frequency approximation step), we must address the impact of their intrinsic bias on our proposed mechanism. Hence, to assess the detection performance (RQ1-RQ3), we conduct 15 repetitions for each configuration, using random hashing coefficients. Moreover, we present the results at a 95% confidence level. In contrast, the mitigation performance evaluation (RQ4) already assumes accurate detection. Thus, it is not necessary to evaluate the effects of the hashing coefficients over multiple repetitions. We expect variability *within* each experiment of the mitigation performance assessment. As an attack progresses, mitigation accuracy changes between different observation windows (OWs). Thus, we report the 95% confidence intervals for the mean of all the measurements in the detection phase. Across all experiments (RQ1-RQ4), we set the smoothing coefficient of the exponentially-weighted moving averages and deviations to $\alpha = 20 \cdot 2^{-8}$.

[3]Available at https://www.github.com/aclapolli/ddosd-cpp.

B. Entropy Estimation Error

Resource constraints of programmable data planes require space- and time-efficient designs. Thus, instead of attempting to calculate exact entropies, we propose estimating these values (Section IV-A2). While estimation reduces the demands for memory space and processing time, it inexorably diminishes accuracy. Such a loss of accuracy can hide traffic anomalies, which would hinder detection performance. Consequently, we must assess the accuracy of our entropy estimates as a function of the count sketch dimensions (RQ1), which are the most crucial factors for determining the accuracy of the sketch-approximated frequencies [20].

In our extended count-sketch implementation, we store each counter in a 32-bit register and its associated observation window (OW) identifier in an 8-bit register. We store the entropy estimate $\hat{H}$ and the entropy norm estimate $\hat{S}$ in 32-bit registers using fixed-point notation with four fractional bits. We populate the longest-prefix match lookup (LPM) table for our pre-computed function, ensuring a maximum absolute error of 2^{-4} for each entry. The resulting LPM table contains a total of 245 TCAM entries of 32 bits each, i.e., 980 bytes.

Figure 5 shows the relative estimation error for each count-sketch depth and width level listed in Table I (first column). By definition, the sketch width (w) is inversely related to the probability of hashing collisions [20]. By following the horizontal axis, we can observe how this factor affects the estimation error: larger widths reduce errors, although this reduction is attenuated until it stabilizes close to 1%. We highlight that the pre-computed function also slightly impacts the relative error, but the plot combines both influences.

Increases in the sketch depth d reduce the probability of obtaining estimates from counters affected by hashing collisions. By examining the error values for a single width, we can

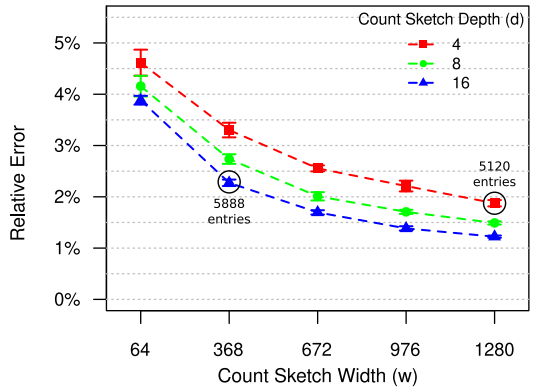


Fig. 5. Relative error of the entropy estimation as a function of the count-sketch width and depth.

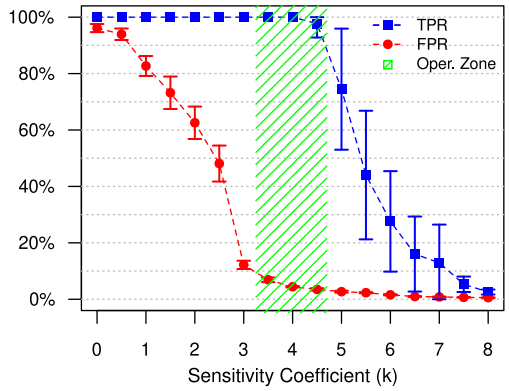


Fig. 6. Impact of the sensitivity coefficient k on the true-positive and false-negative attack detection rates. The area in green highlights the desired operating zone.

observe how the depth affects accuracy. Nevertheless, larger sketch depths require (*i*) processing more hash functions for each packet and (*ii*) more execution steps to calculate the median (see Section IV-A1). We annotate Figure 5 with the total number of sketch entries (5 888 and 5 120) in two specific depths ($d = 16$ and $d = 4$, respectively). These values reveal that, for comparably-sized sketches, increasing the depths does not significantly improve the accuracy of the estimates. Hence, we decide to set $d = 4$ in the subsequent experiments.

C. DDoS Attack Detection Performance

EUCLID allows network operators to configure the sensitivity coefficient (k) in order to obtain a suitable trade-off between the true-positive rate (TPR) and the false-positive rate (FPR) of attack detection. In the detection performance analysis, the TPR refers to the attack phase and indicates the number of OWs in which we detect attacks divided by the number of OWs in which there is an attack. The FPR refers to the pre- and post-attack phases and indicates the number of OWs in which we detect attacks divided by the total number of OWs in the pre- and post-attack phases. Seeking to answer RQ2, we first tune the factor k by observing its effects on the TPR and the FPR (Section V-C1). Then, we study the detection accuracy as related to the attack proportion and the memory usage (Section V-C2)

1) Sensitivity Coefficient Effect: In this experiment, we set the sketch dimensions to $d = 4$ and $w = 1\,280$ (Section V-B). The proportion of malicious packets to the total number of packets during the attack is 5%. Figure 6 presents the TPR and the FPR for attack detection as a function of the sensitivity coefficient k. We can observe that for lower sensitivity coefficients, detection reaches excellent TPRs, i.e., close to 100%. However, the elevated FPRs indicate excessive proportions of false alarms. Following the horizontal axis, as we increase k, both TPR and FPR decrease until the mechanism ceases to generate attack alarms. The false-positive rate decreases from $k = 0$, reaching less than 10% for $k \geq 3.25$. From this point on, the true-positive rate remains close to 100% as long as $k \leq 4.75$. Thus, our mechanism is in its desired operating zone when the sensitivity coefficient k is within [3.25, 4.75] (green hachure). Due to network traffic variability, it may be necessary to adjust k periodically. In our current design, it is the responsibility of the network operator to set and update k to an appropriate level. In future work, we will address this by proposing an automatic self-tuning of the sensitivity coefficient.

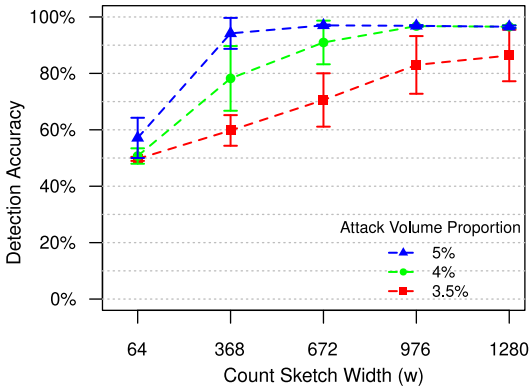


Fig. 7. DDoS attack detection accuracy in terms of memory utilization for different proportions of malicious traffic.

2) DDoS Attack Detection Accuracy: For this analysis, we set the sensitivity coefficient (k) set to 3.5 (which is within the operating range discussed in Section V-C1). We now study the attack detection *accuracy* of our mechanism under various count-sketch widths (which correspond to memory utilization) and proportions of malicious traffic (see Table I).

Figure 7 shows a curve for each attack proportion we consider. As attacks become increasingly aggressive, the detection accuracy reaches progressively higher rates (exceeding 90%). This outcome stems from the more significant entropy anomalies that stronger attacks cause. Along all curves, we observe that the sketch width profoundly influences the detection accuracy. This effect is noticeable even for less intense attacks (3.5%), which EUCLID detects with accuracy higher than 80% for $w \geq 976$. As we increase sketch sizes, entropy estimates become more accurate, which facilitates the detection of subtler attacks.

While enlarging sketches does improve detection accuracy, it also implies an increased memory footprint. We illustrate how to calculate the static random-access memory (SRAM)

requirements as follows. First, recall from Section IV-A3 that we track entropies separately for source and destination IP addresses. Thus, frequency approximation needs *two* sketches. Next, we use the sketch dimensions to obtain the total number of entries across both sketches. Then, we consider that each entry consists of a 32-bit counter and an 8-bit observation window identifier, totaling 40 bits per entry. Finally, we multiply the total number of entries by the entry size. For instance, assuming $d = 4$ and $w = 976$, we obtain $2 \cdot 4 \cdot 976 \cdot 40 \approx 312$ kilobits, i.e., 38.125 kB of SRAM. The footprint just described applies to each 1 Gbps link. Higher data rates demand larger observation windows in order to enable a robust representation of the address frequency distributions. Given that the count-sketch estimation error is proportional both to $1/\sqrt{m}$ (where m is the length of the observation window) and to the ℓ_2 norm of the address frequencies,[4] faster data links require using proportionally larger sketches. Consequently, considering a 24-port 10 Gbps programmable switch [14], we extrapolate the EUCLID memory footprint for detection to 8.93MB, which amounts to 20% of the available SRAM (44.11MB).

D. Comparison With Packet Sampling

Programmable switches can collect fine-grained data about all forwarded packets, which facilitates the deployment of highly-sensitive attack detection mechanisms. In contrast, detection strategies that rely on packet sampling must operate on significantly less data, which limits the detection accuracy. We explore the relation between attack intensity and detection accuracy by comparing EUCLID with an implementation of our detection strategy that receives samples from an sFlow collector (RQ3).

In this section, we perform our experiments using $d = 4$, $w = 1280$, and $k = 4$ (the center of the operating zone discussed in Section V-C1). We assess the sFlow-based mechanism using two different sampling rates: (*i*) 1 : 1 000, which Phaal [54] suggests for a 1 Gbps link, and (*ii*) 1 : 100, in an attempt to improve the detection results. Thus, we have three different scenarios for the assessment: our strategy and the two sFlow-based implementations. To ensure comparable baselines, we set different observation window sizes m for each scenario. For instance, during the time EUCLID processes m packets, the sFlow collector exports only approximately m/1 000, or m/100 packets, depending on the sampling rate. Consequently, we scale m such that the time frames of the three scenarios become approximately equal.

Figure 8 depicts the attack detection accuracy for the three scenarios discussed above under different attack volume proportions. The lower curve indicates that the 1:1 000 sampling rate yields a severely degraded detection performance. At a 1:100 sampling rate, the sFlow-based implementation shows a significantly improved accuracy. Nevertheless, EUCLID outperforms the sFlow approach in every observed attack strength.

We also investigate the attack detection delay by analyzing the timestamps of the packets in our workloads. We use the timestamp of the first malicious packet to indicate the beginning of the attack. We consider the time of detection

[4]We assume the ℓ_2 norm increases proportionally to the traffic rate.

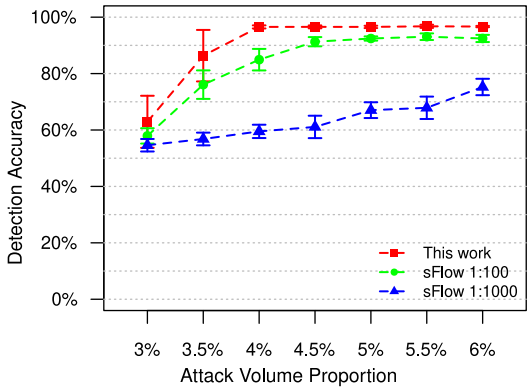


Fig. 8. DDoS attack detection accuracy: comparison with packet sampling approaches.

the timestamp of the last packet of the OW that identifies the entropy anomaly. We observe that for lower attack proportions ($p \leq 4\%$), packet sampling takes several seconds to detect an attack. Under similar conditions, EUCLID detects an attack in a fraction of the time, i.e., a few hundred milliseconds. Such a lower delay may lead to earlier activation of attack mitigation mechanisms, thus potentially preventing service degradation or outage.

E. DDoS Attack Mitigation Performance

Whereas in the previous subsections, we investigate detection accuracy, in this subsection, we analyze the performance of our mitigation strategy. In the mitigation performance analysis, the true-positive rate (TPR) and the false-positive rate (FPR) indicate, respectively, the proportions of *packets* correctly and incorrectly identified as malicious. We underscore that EUCLID was originally designed to handle high-rate volumetric attacks (i.e., the packet rate is expected not to be 'low'). In evaluating the detection and mitigation components of EUCLID, we look for the worst-case/tipping point, which is different in each case. For the detection component, we put it under stress with low proportions of malicious traffic (e.g., 3% to 6%). This is when detection accuracy may degrade (see the lowest curve in Figure 7), which is in line with the literature [55]. Conversely, the mitigation component may reach its limits when the amount of traffic subjected to further inspection/filtering is high (e.g., 20% of malicious traffic). Hence, we set the attack intensity to 20% (i.e., $p = 0.2$). As for the other factor levels, we take into consideration the results from Section V-C regarding count-sketch dimensions (w and d) and sensitivity coefficient (k).

By design (Section IV), EUCLID raises attack alarms only at the end of the observation windows (OWs) in which malicious traffic causes excessive changes in entropy. Consequently, mitigation starts operating only at the beginning of the next OW. For instance, if a detectable incident emerges during OW_i, our countermeasures become active as OW_{i+1} begins. Therefore, we perform the experiments of this section under different observation window (OW) lengths in order to investigate whether it is possible to (*i*) shorten the mitigation delay and (*ii*) reduce the amount of memory required.

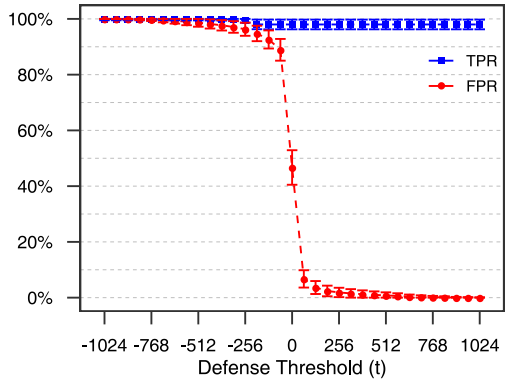


Fig. 9. Effects of the defense threshold t on the true-positive and false-positive packet classification rates.

1) Mitigation Threshold Effect: For an observation window size of $m = 2^{18}$ packets, we measure the effect of different defense thresholds (t) in the interval $[-1\,024;1\,024]$, with 64-packet increments. We consider observation windows from OW_{i+1} to OW_{i+k}, where $i + 1$ identifies the first window after attack detection, and $i + k$ indicates the last window in the attack segment. Finally, we calculate 95% confidence intervals for the mean TPR and FPR during the attack.

Figure 9 shows consistently high true-positive rates, between 98.7% and 100%, for thresholds $t \leq -256$ packets. After this point, for all tested values of t, the TPRs remain elevated, ranging from 96.3% to 99.7%. This result indicates that EUCLID correctly identifies most of the malicious packets in the attack. As a result, by enforcing a security policy (e.g., discarding), our mechanism can prevent over 96% of the spurious traffic from disrupting services at their target. We can also observe that the FPR quickly decreases as t increases. Notably, for $t \geq 768$ packets (approximately 0.25% of the window size), the FPR becomes less than 0.7%. Such a low FPR indicates that the mitigation threshold has near-perfect specificity, i.e., addresses whose observed frequency variations fall *below* the mitigation threshold can be safely assumed legitimate. Since the proportion of legitimate packets incorrectly classified as suspects is negligible, applying a mitigation policy to such packets is unlikely to cause noticeable problems to legitimate users.

2) Observation Window Size Effect: We now investigate the performance of attack mitigation under different observation window (OW) sizes. Measurement fluctuations related to these different OW sizes require adjusting the sensitivity coefficient (k) within the operating range (Table I, last column).

For all three factor levels of m, we observe similar outcomes, i.e., the TPR remains over 90%, and the FPR quickly decreases as the mitigation threshold t grows. Moreover, the t value that yields near-zero FPRs varies with the OW size. We summarize these results in Table II. These findings indicate that reducing the observation window length does not hinder the accuracy of our mitigation mechanism.

Considering the 1Mpps average packet rate of our workload, an observation window with 2^{18} packets corresponds to approximately 250 ms of traffic. By reducing the OW length to 2^{14} packets, each OW will take close to 15 ms. Such a reduction can further reduce the attack mitigation delay. Since the memory space needed for frequency approximation is proportional to the size of the OW (Section V-C2), we can also reduce the amount of SRAM required for detection and mitigation.

TABLE II
EFFECTS OF THE OBSERVATION WINDOW SIZE m

OW Size (m)	2^{14}	2^{16}	2^{18}
Threshold (t)	96	384	768
TPR (%)	[97.45, 98.34]	[97.04, 98.81]	[96.26, 99.74]
FPR (%)	[0.00, 0.08]	[0.00, 0.05]	[0.00, 0.63]
Defense Delay	$\approx$ 16 ms	$\approx$ 64 ms	$\approx$ 256 ms

3) Effects on Traffic Latency: Accurate measurements with hardware-based experiments of the latency introduced by EUCLID (actually, by any P4 design) will lead to results that vastly vary depending on the specific devices employed. Aiming to provide the reader with a more general analysis, we determine a theoretical upper bound for latency. To do this, we consider that our $\mathrm{P4}_{16}$ prototype targets the RMT model implementation by Bosshart *et al.* [14], whose switch chip: (*i*) has a 1GHz operating frequency; (*ii*) parses packet headers in a single cycle; (*iii*) has 32 match-action stages in each pipeline (ingress and egress); and (*iv*) performs each match-action in a single cycle.[5] EUCLID uses only the *ingress* parser, pipeline, and deparser processing blocks[6] and does not require packet recirculation. Therefore, a worst-case scenario incurs the forwarding delay resulting from the sequential processing of 34 stages (32 match-action + 2 [de]parser), at 1ns per stage, i.e., 34ns. This analysis can be readily performed for alternative targets, but it is reasonable to expect average latencies in the order of tens of nanoseconds (including in experimental studies).

F. Applicability and Limitations

Different DDoS attacks require distinct defense strategies. Our evaluation has so far indicated that EUCLID is effective against volumetric flooding attacks coming, e.g., from botnets (i.e., high source address entropy and low destination address entropy). Nevertheless, we can still reflect on our proposed solution's behavior under scenarios for which it was not designed originally, such as semantic, increasing rate, and amplification-based attacks.

Semantic Attacks: Semantic attacks are also known as protocol exploitation or state exhaustion attacks [26]. These attacks are generally low-rate and work by intentionally allowing otherwise legitimate transactions to time out, thus withholding and depleting resources on the target systems. Some notable instances are the HTTP-based slow DDoS attacks discussed in [56], such as Slow HTTP Headers (Slowloris), Slow HTTP POST (RUDY), and Slow Read. Defending against this type of attack requires tracking state information (at the transport or application layer) for an extended time. EUCLID, by design, observes network-layer traffic patterns; thus, it does not cover

[5]Parallelism allows multiple match-actions in a single cycle.
[6]We only use the egress processing blocks to output diagnostic data.

semantic attacks. Defending against them would require a different type of solution beyond our work scope.

Increasing-Rate Attacks: In certain brute-force attacks, the attack rate starts low and gradually increases, in an attempt to manipulate the baseline traffic characterization model, thus evading detection [21]. To improve the robustness of the traffic model under such "slow-start" attacks, the network operator can adjust the sensitivity coefficient (Section IV-A3) and the cooldown period (Section IV-B1). More specifically, he/she is expected to tune these parameters more conservatively, making EUCLID spend more time in the defense states (in which we do not update the traffic model). Since the packet classification FPR is remarkably low, we do not anticipate traffic overhead issues from the more frequent activation of the mitigation mechanism.

Amplification-Based Attacks: Attacks that rely on amplification strategies would also cause entropy anomalies. Differently from botnet-originated campaigns, however, amplification causes sudden, pronounced decreases in both source and destination entropy measurements. As we discussed in Section III-B, detection occurs whenever at least one of the entropy measurements deviates from the model. Consequently, a quick drop in destination entropy would make the Anomaly Detection block issue an alarm and engage the mitigation mechanisms. Regarding mitigation, given that the IP address frequency variations caused by reflection attacks are still anomalous, we can readily classify and treat malicious packets accordingly.

Fluctuations in the Number of Flows: A trace with large fluctuations in the number of flows would induce relevant entropy variations. The scale of the entropy variations influences how strict the traffic model is. If we train the model with traces that exhibit large fluctuations, detection will also require large anomalies. It is possible that flow accounting could add details that might be useful to improve our mechanism accuracy. However, our experiments show that the attacker's behavior is sufficiently disruptive to cause detectable entropy anomalies within our attack model. Thus, we advocate that using IP addresses is good enough for detection. Our tuning parameters (e.g., the sensitivity coefficient) can be adjusted to what is expected in a given scenario, allowing an operator to obtain adequate attack detection FPR and TPR values. Moreover, implementing flow accounting would require adding extra memory for bookkeeping. Exploring this compelling research avenue and assessing whether such a change would improve accuracy is left as future work.

VI. LESSONS LEARNED AND INSIGHTS

While data plane programmability brings flexibility to packet forwarding, implementing this paradigm on hardware is a challenge. Obtaining an adequate trade-off between high performance and reasonable production costs is paramount. Modern programmable data planes reach this goal by carefully delimiting reconfigurability to a core repertoire of primitives related to packet forwarding (e.g., header parsing and match-action tables). Moreover, these data planes do not implement several popular programming constructs (e.g., repetition statements, stacks, and non-trivial mathematical operations). Additionally, since the amount of memory directly influences chip area and power consumption, packet switches provide relatively small sizes of SRAM and TCAM. On the one hand, these hardware design choices help to prevent stalls in the packet processing pipeline as well as prohibitively complex hardware layouts. On the other hand, algorithms for programmable switches require particularly careful design and implementation. We detail the most important lessons learned from the design of EUCLID in the remainder of this section.

a) Limited syntactic expressiveness requires careful programming practices: P4-programmable switches have limited support for procedure definition and invocation (i.e., match-action tables are not as flexible as the function call and return instructions). Consequently, code reuse becomes challenging, which makes the implementation process more intricate. Intuitively, a way to work around this limitation is to write tools to generate P4 code automatically. However, perhaps it would be better to enrich the P4 language with standardized higher-level constructs that the compiler or preprocessor could turn into native P4 code. This strategy could also promote the widespread adoption and distribution of libraries with stable implementations of well-established building blocks (e.g., Bloom filters, count-sketches, and algorithms to approximate non-trivial mathematical functions).

b) Event-driven processing can compensate for the absence of iterative procedures: In a general way, all real-time systems have stringent time budgets. This characteristic profoundly influences the architecture of programmable data planes, which ultimately prompts algorithm redesign. In this work, this observation emerges when addressing three design challenges: (*i*) the summation of the individual address frequency terms required by Equation (1), (*ii*) the need to reset sketch counters between observation windows in order to avoid outdated values, and (*iii*) the necessity to copy data between sketches. It is unfeasible to iterate over entire sketches for every single incoming packet since the resulting overhead would exceed the time budget that line-rate packet processing requires. Thus, we need to redesign iterative procedures as event-driven strategies in which every packet arrival triggers the execution of smaller, tractable steps. EUCLID handles challenge (*i*) by gradually accumulating the entropy norm variation as each packet arrives at the switch. We tackle challenge (*ii*) by augmenting the sketch counters with an observation window (OW) identifier W_{ID} and modifying the count sketch operations such that accesses to outdated counters produce an automatic reset followed by an update to the current window ID. Thus, we defer the resetting of each cell until they are necessary. Similarly, we approach challenge (*iii*) by placing copy operations close to counter updates, so that, right before resetting counters, we can perform the required copies we discussed in Section IV-B2.

c) The number of pipeline stages in the data plane limits the size of multi-step procedures: The time budget of programmable data planes also constrains the maximum length of the code to deploy on the switch. As a result, attempts to manage the absence of iteration (e.g., by resorting to loop unrolling) and subroutines (e.g., by automatic generation of

repeated code segments) might not always be a viable strategy. Thus, algorithm design needs to take into consideration the space requirements for the resulting code.

d) Pre-computed functions can address the lack of non-elementary mathematical functions: Current programmable switches do not directly implement operations such as divisions, exponentiations, and logarithms. Thus, we need to write algorithms to approximate these functions in terms of the primitives already available on them. We can handle this challenge by numerically analyzing the function signature concerning its domain and image bounds. We do this to identify opportunities for compact representations tailored to the use-case at hand. In our work, there is the need to calculate updates to the entropy norm terms, which depend on the binary logarithm (Equation (10)). The entropy norm update function has strictly-bounded intervals for both domain (frequencies ranging from one to the OW size) and image (Figure 4). As a result, it is possible to build a memory-efficient longest prefix match (LPM) table by aggregating domain entries with close values while ensuring enough accuracy for our purposes. Alternatively, for the general case, one can use numeric algorithms explicitly developed for in-switch execution [47].

e) Floating-point arithmetic may not be essential to operate numbers whose precision and range are strictly constrained: As traditional packet forwarding requires only integer arithmetic, switches typically lack floating-point instructions and registers. Nevertheless, integer arithmetic can operate on fractional numbers given a fixed-point representation. Throughout this work, we express non-integer quantities in fixed-point notation using scaling factors from 2^3 to 2^{18}. These numbers are the smoothing and sensitivity coefficients, the entropy norms, and the indices of central tendency and dispersion. Our evaluation (Section V) shows that fixed-point representation is sufficiently accurate both to detect and to mitigate DDoS attacks.

f) The absence of dynamic memory allocation in the data plane limits the flexibility of the self-tuning of the mechanism: EUCLID has several tuning parameters (i.e., α, k, m, and t) that the network operator can modify at runtime by updating register values. However, changes in parameters that dictate the memory size of data structure (i.e., the sketch dimensions d and w) are not straightforward. Since modern programmable data planes do not provide dynamic memory management, the compiler is responsible for allocating memory statically. Thus, changes in memory layout require reinstalling the P4 program, which is a disruptive operation. In this scenario, enhancing the flexibility of self-tuning beyond simple changes in register values demands the investigation of novel P4 constructs.

VII. CONCLUSION

In this article, we proposed EUCLID, a novel real-time DDoS attack detection and mitigation mechanism that can be executed entirely in a programmable data plane. This work shows that our P4-based design has the potential to meet increasingly strict performance requirements in high-volume networks. As another significant contribution, we shared lessons learned during the design, implementation, and evaluation of EUCLID, hoping these insights may be helpful for future research on programmable networks. We also share with the community the source code of our prototype implementation and of our analysis toolkit, which can be found at https://www.github.com/asilha/euclid. The datasets used in the evaluation can be obtained from CAIDA [52], [53].

Our experimental evaluation indicates that EUCLID can detect and mitigate the effects of DDoS outbreaks quickly and accurately. In a link with a traffic rate of one million packets per second, EUCLID detects attacks and launches its mitigation mechanism within 250 ms. Attack detection is over 90% accurate, correctly signaling most DDoS incidents while keeping the proportion of false alarms below 10%. We observed that the detection accuracy of our mechanism is superior to that of an approach based on packet sampling. We estimate that deploying detection on all interfaces of a 10 Gbps switch requires a total of 9 MB of SRAM, well within the capacity of existing programmable data plane targets. Attack mitigation correctly identifies as suspects more than 96% of malicious packets, with a false-positive rate (FPR) smaller than 1%. EUCLID can effectively steer the attack traffic away from its intended target, thus preventing service outage. The low FPR ensures that legitimate users can still access the protected service. Our mitigation components require an additional 375 kB of SRAM per 10 Gbps link.

As future work, we plan on providing EUCLID with dynamic self-tuning of the sensitivity coefficient and of the mitigation threshold in response to traffic patterns and trends. Such self-tuning may build on strategies ranging from statistical methods to machine learning techniques. On a different research path, our roadmap also includes the investigation of novel P4 constructs to make it possible to modify data structure sizes without disrupting network traffic.

REFERENCES

[1] R. Hummel, C. Hildebrand, H. Modi, and G. Sockrider, "NETSCOUT threat intelligence report with key findings from the 15th annual worldwide infrastructure security report (WISR)," Netscout Syst. Inc., Westford, MA, USA, Rep. SECR_001_EN-2001, Feb. 2020. Accessed: Mar. 20, 2020. [Online]. Available: https://www.netscout.com/sites/default/files/2020-02/SECR_001_EN-2001_Web.pdf

[2] T. Shani. (2019). *Updated: This DDoS Attack Unleashed the Most Packets Per Second Ever. Here's Why That's Important*. Accessed: Mar. 20, 2020. [Online]. Available: https://bit.ly/33PR7zM

[3] C. Crane. (2019). *The Largest DDoS Attacks in History*. [Online]. Available: https://www.thesslstore.com/blog/largest-ddos-attack-in-history/

[4] S. Kottler. (Mar. 2018). *February 28th DDoS Incident Report. GitHub*. [Online]. Available: https://githubengineering.com/ddos-incident-report/

[5] S. Hilton. (Oct. 2016). *Dyn Analysis Summary Of Friday October 21 Attack. Oracle Dyn*. [Online]. Available: https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/

[6] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, "Millions of targets under attack: A macroscopic characterization of the DoS ecosystem," in *Proc. Internet Meas. Conf. (IMC)*, 2017, pp. 100–113. [Online]. Available: https://doi.org/10.1145/3131365.3131383

[7] A. Chadd, "DDoS attacks: Past, present and future," *Netw. Security*, vol. 2018, no. 7, pp. 13–15, Jul. 2018. [Online]. Available: https://doi.org/10.1016/S1353-4858(18)30069-2

[8] "Cloud in the crosshairs: 14th annual worldwide infrastructure security report (WISR)," Netscout Syst. Inc., Westford, MA, USA, Rep. SECR_005_EN-1901, Mar. 2019. Accessed: Mar. 20, 2020. [Online]. Available: https://www.netscout.com/sites/default/files/2019-03/SECR_005_EN-1901%E2%80%93WISR.pdf

[9] N. Feamster, J. Rexford, and E. Zegura, "The road to SDN: An intellectual history of programmable networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014. [Online]. Available: http://doi.acm.org/10.1145/2602204.2602219

[10] P. Phaal, S. Panchen, and N. McKee. (Sep. 2001). *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. Internet Requests for Comments*. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3176.txt

[11] B. Claise. (Oct. 2004). *Cisco Systems NetFlow Services Export Version 9. Internet Requests for Comments*. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3954.txt

[12] The Open Networking Foundation. (Mar. 2015). *OpenFlow Switch Specification Version 1.5.1*. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[13] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2013, pp. 73–78. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491196

[14] P. Bosshart *et al.*, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486011

[15] A. Gupta, R. Birkner, M. Canini, N. Feamster, C. Mac-Stoker, and W. Willinger, "Network monitoring as a streaming analytics problem," in *Proc. 15th ACM Workshop Hot Topics Netw. (HotNets)*, 2016, pp. 106–112. [Online]. Available: http://doi.acm.org/10.1145/3005745.3005748

[16] S. Narayana *et al.*, "Language-directed hardware design for network performance monitoring," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 85–98. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098829

[17] J. Boite, P. A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "StateSec: Stateful monitoring for DDoS protection in software defined networks," in *Proc. IEEE Conf. Netw. Softw. (NetSoft)*, Jul. 2017, pp. 1–9. [Online]. Available: https://doi.org/10.1109/NETSOFT.2017.8004113

[18] A. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time DDoS attack detection to programmable data planes," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manag. (IM)*, Apr. 2019, pp. 19–27. [Online]. Available: https://ieeexplore.ieee.org/document/8717869

[19] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[20] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Automata, Languages and Programming* (Lecture Notes in Computer Science), vol. 2380, P. Widmayer, S. Eidenbenz, F. Triguero, R. Morales, R. Conejo, and M. Hennessy, Eds. Berlin, Germany: Springer, Jul. 2002, pp. 693–703, doi: 10.1007/3-540-45465-9_59.

[21] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, 2004. [Online]. Available: https://doi.org/10.1145/997150.997156

[22] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems," *ACM Comput. Surveys*, vol. 39, no. 1, p. 42, 2007. [Online]. Available: https://doi.org/10.1145/1216370.1216373

[23] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 2046–2069, 4th Quart., 2013. [Online]. Available: https://doi.org/10.1109/SURV.2013.031413.00127

[24] N. Hoque, M. H. Bhuyan, R. C. Baishya, D. K. Bhattacharyya, and J. K. Kalita, "Network attacks: Taxonomy, tools and systems," *J. Netw. Comput. Appl.*, vol. 40, pp. 307–324, Apr. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804513001756

[25] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in DDoS attacks: Trends and challenges," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2242–2270, 4th Quart., 2015. [Online]. Available: https://doi.org/10.110910.1109/COMST.2015.2457491

[26] R. Swami, M. Dave, and V. Ranga, "Software-defined networking-based DDoS defense mechanisms," *ACM Comput. Surveys*, vol. 52, no. 2, p. 36, Apr. 2019. [Online]. Available: https://doi.org/10.1145/3301614

[27] M. Zhang *et al.*, "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Proc. NDSS*, Jan. 2020, p. 28. [Online]. Available: https://bit.ly/2vZviRE

[28] M. Bonfim, M. Santos, K. Dias, and S. Fernandes, "A real-time attack defense framework for 5G network slicing," *Softw. Pract. Exp.*, vol. 50, no. 7, pp. 1228–1257, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2800

[29] Y. Xu and Y. Liu, "DDoS attack detection under SDN context," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2016, pp. 1–9. [Online]. Available: https://doi.org/10.1109/INFOCOM.2016.7524500

[30] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, "OpenState: Programming platform-independent stateful OpenFlow applications inside the switch," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 44–51, Apr. 2014. [Online]. Available: http://doi.acm.org/10.1145/2602204.2602211

[31] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *Proc. 10th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, Lombard, IL, USA, 2013, pp. 29–42. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/yu

[32] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, 2016, pp. 101–114. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934906

[33] T. Yang *et al.*, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2018, pp. 561–575. [Online]. Available: http://doi.acm.org/10.1145/3230543.3230544

[34] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: Methods, evaluation, and applications," in *Proc. 3rd ACM SIGCOMM Conf. Internet Meas.*, 2003, pp. 234–247. [Online]. Available: https://doi.org/10.1145/948205.948236

[35] C. Wang, T. T. N. Miu, X. Luo, and J. Wang, "SkyShield: A sketch-based defense system against application layer DDoS attacks," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 3, pp. 559–573, Mar. 2018. [Online]. Available: https://doi.org/10.1109/TIFS.2017.2758754

[36] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2013, pp. 413–424. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516684

[37] Y. Afek, A. Bremler-Barr, and L. Shafir, "Network anti-spoofing with SDN data plane," in *Proc. IEEE INFOCOM Conf. Computer Commun.*, May 2017, pp. 1–9. [Online]. Available: https://doi.org/10.1109/INFOCOM.2017.8057008

[38] K. Tavares and T. Ferreto, "DDoS on sketch: Spoofed DDoS attack defense with programmable data planes using sketches in SDN," in *Proc. Anais do Simpsio Brasileiro de Redes de Computadores e Sistemas Distribudos*, vol. 37, 2019, pp. 805–819.

[39] F. Paolucci, F. Civerchia, A. Sgambelluri, A. Giorgetti, F. Cugini, and P. Castoldi, "P4edge node enabling stateful traffic engineering and cyber security," *IEEE/OSA J. Opt. Commun. Netw.*, vol. 11, no. 1, pp. A84–A95, Jan. 2019. [Online]. Available: http://jocn.osa.org/abstract.cfm?URI=jocn-11-1-A84

[40] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677403001913

[41] J. Xing, W. Wu, and A. Chen, "Architecting programmable data plane defenses into the network with FastFlex," in *Proc. 18th ACM Workshop Hot Topics Netw. (HotNets)*, 2019, pp. 161–169. [Online]. Available: https://doi.org/10.1145/3365609.3365860

[42] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 623–656, Jul.–Oct. 1948.

[43] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 217–228, Aug. 2005. [Online]. Available: http://doi.acm.org/10.1145/1090191.1080118

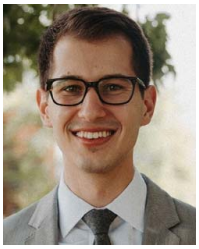
[44] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "An empirical evaluation of information metrics for low-rate and high-rate DDoS attack detection," *Pattern Recognit. Lett.*, vol. 51, pp. 1–7, Jan. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016786551400244X

[45] S. W. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 1, no. 3, pp. 239–250, 1959. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00401706.1959.10489860

[46] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proc. Symp. SDN Res. (SOSR)*, 2017, pp. 164–176. [Online]. Available: http://doi.acm.org/10.1145/3050220.3063772
[47] D. Ding, M. Savi, and D. Siracusa, "Estimating logarithmic and exponential functions to track network traffic entropy in P4," in *Proc. IEEE/IFIP Netw. Oper. Manag. Symp. (NOMS)*, Apr. 2020, pp. 1–9.
[48] G. Cormode, "Sketch techniques for approximate query processing," *Found. Trends Databases*, vol. 4, nos. 1–3, pp. 1–294, 2011. [Online]. Available: http://dx.doi.org/10.1561/1900000004
[49] Barefoot Networks. (2020). *Tofino: World's Fastest P4-Programmable Ethernet Switch ASICs*. [Online]. Available: https://barefootnetworks.com/products/brief-tofino/
[50] The P4Language Consortium. (2020). *BMv2. P4Language Consortium*. [Online]. Available: https://github.com/p4lang/behavioral-model
[51] S. Chole *et al.*, "DRMT: Disaggregated programmable switching," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 1–14. [Online]. Available: https://doi.org/10.1145/3098822.3098823
[52] CAIDA. (2016). *The CAIDA UCSD Anonymized Internet Traces 2016*. [Online]. Available: http://www.caida.org/data/passive/passive_2016_dataset.xml
[53] CAIDA. (2007). *The CAIDA UCSD DDoS Attack 2007 Dataset*. [Online]. Available: http://www.caida.org/data/passive/ddos-20070804_dataset.xml
[54] P. Phaal. (Jun. 2009). *sFlow: Sampling Rates*. [Online]. Available: https://blog.sflow.com/2009/06/sampling-rates.html
[55] Y. Xiang, K. Li, and W. Zhou, "Low-rate DDoS attacks detection and traceback by using new information metrics," *IEEE Trans. Inf. Forensics Security*, vol. 6, no. 2, pp. 426–437, Jun. 2011. [Online]. Available: https://doi.org/10.1109/TIFS.2011.2107320
[56] N. Muraleedharan and B. Janet, "Behaviour analysis of HTTP based slow denial of service attack," in *Proc. Int. Conf. Wireless Commun. Signal Process Netw. (WiSPNET)*, 2017, pp. 1851–1856. [Online]. Available: https://doi.org/10.1109/WiSPNET.2017.8300082

Alexandre da Silveira Ilha received the B.Sc. degree in computer science and a specialization certificate in cyber security from the Federal University of Rio Grande do Sul, Brazil, in 2006 and 2019, respectively, where he is currently pursuing the M.Sc. degree in computer science, investigating security mechanisms that leverage data plane programmability to improve network monitoring and intrusion detection systems performance. His objective is to advance the state-of-the-art of network security to overcome increasingly more challenging scenarios.

Ângelo Cardoso Lapolli received the B.Sc. degree in computer engineering and the M.Sc. degree in computer science from the Federal University of Rio Grande do Sul, Brazil, in 2014 and 2019. His main research interest is network management, with emphasis on the design of security mechanisms founded on software-defined networking and data plane programmability.



Jonatas Adilson Marques received the B.Sc. degree in computer science and the M.Sc. degree in applied computing from Santa Catarina State University, Brazil, in 2015 and 2017, respectively. He is currently pursuing the Ph.D. degree with the Institute of Informatics, Federal University of Rio Grande do Sul, Brazil, doing research in network management, with a focus on software-defined networking and network programmability. His goal is to conceive techniques and systems that make use of the new capabilities of programmable networks to improve how networks are managed and enable them to support both emerging and future Internet applications.

Luciano Paschoal Gaspary (Senior Member, IEEE) received the Ph.D. degree in computer science from UFRGS in 2002, where he is currently the Deputy Dean and an Associate Professor with the Institute of Informatics. From 2008 to 2014, he worked as the Director of the National Laboratory on Computer Networks and from 2009 to 2013, he was the Managing Director of the Brazilian Computer Society. He has authored more than 200 full papers published in leading peer-reviewed publications and has a history of dedication to research activities such as the organization of scientific events, participation in the TPC of relevant symposia, and participation as an editorial board member of various journals. He has been involved in various research areas, mainly computer networks, network management, and computer system security. He is currently the Editor-in-Chief for *Journal of Network and Systems Management* (Springer). Further information regarding his biography can be found at https://www.inf.ufrgs.br/∼paschoal/.