Offloading Real-time DDoS Attack Detection to Programmable Data Planes

Ângelo Cardoso Lapolli, Jonatas Adilson Marques, and Luciano Paschoal Gaspary
Institute of Informatics, Federal University of Rio Grande do Sul - Brazil
{aclapolli, jamarques, paschoal}@inf.ufrgs.br

***Abstract*—In recent years, Distributed Denial-of-Service (DDoS) attacks have escalated both in frequency and traffic volume, with outbreaks reaching rates up to the order of terabits per second and compromising the availability of supposedly highly resilient infrastructure (e.g., DNS and cloud-based web hosting). The reality is that existing detection solutions resort to a combination of mechanisms, such as packet sampling and transmission of gathered data to external software, which makes it very difficult (if at all possible) to reach a good compromise for accuracy (higher is better), resource usage footprint, and latency (lower is better). Data plane programmability has emerged as a promising approach to help meeting these requirements as forwarding devices can be configured to execute algorithms and examine traffic at line rate. In this paper, we explore P4 primitives to design a fine-grained, low-footprint, and low-latency traffic inspection mechanism for real-time DDoS attack detection. Our proposal – the first to be fully in-network – contributes to shed light on the challenges to implement sophisticated security logic on forwarding devices given that, to operate at high throughput, the inspection (and overall processing) of packets is subject to a small time budget (dozens of nanoseconds) and limited memory space (in the order of megabytes). We evaluate the proposed mechanism using packet traces from CAIDA. The results show that it can detect DDoS attacks entirely within the data plane with high accuracy (98.2%) and low latency ($\approx$250 ms) while keeping device resource usage low (dozens of kilobytes in SRAM per 1 Gbps link and a few hundred TCAM entries).**

I. Introduction

Despite consistent efforts towards effective detection and mitigation mechanisms, Distributed Denial-of-Service (DDoS) attacks remain among the top networking security concerns as outbreaks escalate both in frequency and traffic volume [1]. Reports of recent attacks targeting Dyn [2] and GitHub [3] reveal peak rates of up to the order of terabits per second compromising the availability of (supposedly) scalable and resilient infrastructure. Given this trend, we should expect events like these to get even worse in the future [1].

Existing defensive mechanisms typically rely on standardized monitoring primitives such as packet sampling (e.g., sFlow [4]) and flow-based accounting (e.g., NetFlow [5] and OpenFlow [6]). However, these primitives present significant overheads regarding packet processing and resource utilization to provide fine-grained traffic visibility. While packet sampling conveys information from a reduced set of packets to keep a reasonable load in terms of CPU processing and network management traffic [7], flow-based accounting is limited to aggregated volume metrics due to elevated memory footprint [8]. Resulting from these limitations, we advocate that the existing tooling for monitoring falls short in either *accuracy* or *resource usage* when it comes to DDoS attack detection. Furthermore, these approaches are subject to a long control loop, resulting in non-negligible detection *latency*.

As a promising alternative to these issues, the emerging concept of *data plane programmability* offers flexibility to readily implement novel in-switch packet processing algorithms [9]. These algorithms assume a packet stream as input and are modeled as a pipeline of elementary primitives, memory accesses, and table lookups. Human operators are thus able to define monitoring functions and delegate them to forwarding devices across the whole network. This still relatively unexplored concept has the potential of enabling all packets of a stream to be examined without processing/communication overheads and achieving low-latency anomaly detection. Yet, to operate at line rate on high-speed links, this processing is constrained to a small time budget (dozens of nanoseconds) and a limited memory space (e.g., $\approx$50 MB SRAM and $\approx$5 MB TCAM) [10].

Meeting the aforementioned constraints is a difficult challenge that limits the scope of the existing data plane monitoring solutions. For example, Sonata [11] and Marple [12] take advantage of data plane programmability to configure adaptive filters which determine packet streams to be forwarded to and examined by the control plane. This approach requires continuous communication incurring in non-negligible network usage and potentially high attack detection latency. StateSec [13], in turn, is a DDoS attack detection mechanism fully implemented on Software-Defined Networking (SDN)/OpenFlow-based forwarding devices. It extends the *match/action* table structure and semantics to keep track of harmful packet exchange patterns. Nevertheless, it does not take into account the requirement of scaling to run at line rate on high-throughput hardware packet processors.

In this paper, we give a consistent step towards *in-network*, programmable network security. We explore a promising data plane programming technology, namely P4 [9], to design and implement a mechanism[1] to perform *low-latency, fine-grained traffic inspection* for *real-time DDoS attack detection*. In contrast to existing solutions in the context of SDN, the proposed mechanism is fully implementable on forwarding devices. It comprises a processing pipeline to estimate the entropies of

978-3-903176-15-7 © 2019 IFIP

[1]Available at https://github.com/aclapolli/ddosd-p4

both source and destination IP addresses of incoming packets. The entropy measurements are used to both characterize the traffic and calculate anomaly detection thresholds (as functions of a parameterizable sensitivity coefficient). In order to meet the strict time and memory constraints of forwarding devices, we approximate the frequencies of distinct IP addresses through specially tailored count sketches [14]. Further, compute-intensive arithmetic functions are solved with the aid of a memory-optimized longest-prefix match (LPM) lookup table. Based on realistic datasets of legitimate traffic and DDoS attacks, we assess the entropy estimation error and evaluate the detection performance in terms of accuracy and resource consumption. We also compare the effectiveness and efficiency (i.e., latency) of the proposed mechanism with those of the "de facto" approaches.

The primary research contributions of this paper are threefold. First, we stress data plane programmability primitives (in this work, of P4) – known for their limited functionality – to design a reasonably sophisticated in-network DDoS attack detection mechanism. Second, we demonstrate, through an extensive evaluation, the performance benefits that security mechanisms can reap from a data plane-based design. Third, we discuss challenges and present insights associated with the development of security mechanisms in the data plane that can be valuable for new research initiatives in the area.

The remainder of this paper is structured as follows. In Section II, we discuss recent work related to DDoS attack detection and network monitoring on programmable data planes. In Section III, we describe the proposed DDoS attack detection mechanism. In Section IV, we present the evaluation methodology and results. In Section V, we discuss the major lessons learned in the process of designing the proposed mechanism. In Section VI, we conclude the paper with final remarks and perspectives for future work.

II. Related Work

DDoS attack features have been extensively investigated for outbreak detection and mitigation [15]. Nonetheless, real-time security solutions are limited by the monitoring functionality currently implemented on most forwarding devices, in which accuracy necessarily translates into high overheads [8]. It is in this context that programmable data plane-based approaches emerge as promising alternatives, being subject of consistent research work concerning, for example, scalable in-network packet processing models and fine-grained traffic measurement capability. Next, we review some of the most prominent investigations in the area.

Based on SDN/OpenFlow, Xu and Liu [16] propose methods to detect DDoS attacks and identify their sources and victims. On top of the control plane, a software application classifies flows regarding their volume and rate asymmetry through an unsupervised learning algorithm. This traffic data is collected from the counters associated to flow table entries (in forwarding devices). Since the number of entries is limited, their aggregation granularity is adaptively changed to enable zooming into abnormal traffic patterns. This process is iterative and highly dependent on the control plane, introducing non-negligible latency (in the order of several seconds) to the detection of an ongoing attack.

To offload monitoring functions to the data plane, StateSec [13] is a DDoS attack detection mechanism based on in-switch processing capabilities. StateSec is based on an OpenFlow extension in which flow tables can be used to specify finite-state machines for packet processing [17]. The detection results from entropy analysis of both source/destination IP addresses and transport-layer ports. These metrics along with their mean and standard deviation are supposed to be calculated within the data plane. However, the mechanism requires a flow table entry for every distinct observed IP/port 4-tuple value, which implies unbounded table utilization. Furthermore, it does not elaborate on how to measure entropy while meeting the time budget to operate on high-throughput forwarding devices.

Advancing from more traditional SDN/OpenFlow-based measurement approaches to ones in which accounting is fully delegated to the data plane, OpenSketch [18], UnivMon [19], and Elastic sketch [20] provide flexible hash table-based designs that enable network operators to implement a wide range of measurement tasks in the data plane. The forwarding devices are responsible for maintaining sets of hash tables (named *sketches*) with summarized up-to-date traffic counters. The control plane periodically collects this data for further processing. As a result, these solutions achieve high generality and accuracy in traffic measurement. However, they are subject to a trade-off between the data polling rate (which is directly related to anomaly detection latency) and network overhead due to additional management traffic.

In an attempt to offload additional monitoring logic to the data plane, Sonata [11] allows operators to define packet stream filtering queries. These queries are executed on programmable forwarding devices so that only the traffic of interest is sent to external stream processors. Packet headers are abstracted as tuples of field values, which – in addition to be filtered – can be sampled in the data plane. Based on these abstractions, network operators can optimize packet sampling to detect priorly known anomalous traffic patterns, but potentially missing novel attack strategies. Following a similar concept, Marple [12] is a language for expressing monitoring queries that are compiled to target programmable forwarding devices. It enables in-network execution of functions over aggregation of packets backed by a new key-value store primitive. Despite providing forwading devices with the ability to measure traffic features, the inspection of such metrics is still delegated to external servers (again, incurring additional detection latency).

The area of in-network security management is flourishing, with the potential to allow operators to devise novel attack detection mechanisms within a much shorter design and deployment cycles. The aforementioned proposals represent consistent steps towards devising mechanisms to be executed in the data plane, but (*i*) resort to considerable communication with external controllers (delaying the detection of attacks and leading to a high network utilization) and (*ii*) use coarse-

grained measurements to cope with the massive amount of data traversing high-speed links (degrading accuracy). Our proposed mechanism goes a significant step further than previous work by enabling DDoS attack detection entirely within the data plane. Our design explores data plane programmability functionality to its limit and achieves *accuracy*, *low intrusiveness*, and *timeliness*, as we describe next.

III. Our Design for In-Network DDoS Detection

In this section, we introduce the proposed in-network DDoS attack detection mechanism. First, in Section III-A, we describe both the attack scenario considered and the threat model. Next, in Section III-B, we overview the foundations of the anomaly detection strategy. Then, in Section III-C, we detail the packet processing pipeline devised to run at line rate on programmable forwarding devices.

A. Attack Scenario and Threat Model

The term distributed denial-of-service comprises a multitude of attack strategies to degrade or disrupt external facing services. In this work, we assume an attacker capable of coordinating globally distributed hosts to send illegitimate service requests to a single victim. These requests may either saturate the victim's network with high traffic volume or exploit a specific protocol semantic vulnerability to consume the computing resources of the target server. The attacker uses spoofed IP addresses to hinder the characterization and detection of the attack packets.

In the aforementioned situation, the spread of attacking hosts among independent administrative domains imposes challenges to source-based detection mechanisms, because it is hard to coordinate security efforts across administrative borders. At the other extremity of the attack, i.e., the victim's infrastructure, the malicious traffic is aggregated and prominent for detection, but it may have already saturated both in-path and local resources. Thus, our proposed mechanism is expected to be deployed at the Autonomous Systems (ASs) closest to the victim as they benefit from a privileged traffic view and high-throughput links to timely uncover and deter even voluminous outbreaks without exhausting their resources.

In order to detect attacks and enable mitigation before reaching lower-capacity links, our mechanism should run on nodes peering with other ASs. P4 allows a flexible deployment and parameterization of our mechanism on top of programmable hardware nodes, granting the processing power to keep up with high packet rates.

B. Detection Strategy Foundations

Given that the strict time and memory constraints for high-rate in-network packet processing translate to limited programming primitives, it is paramount to resort to a simple, yet powerful detection strategy. We assume DDoS attacks characterized by a large number of hosts converging traffic to one or few victims [21] in which case the source and destination IP addresses distributions tend to deviate from the legitimate pattern in the presence of malicious activity. In this regard, we design our mechanism based on the calculation of the Shannon entropy [22], which is recognized as a reliable method to identify such deviations accurately [23], [24].

Considering X the set of IP addresses within a total of m packets, and $f_0, f_1, ..., f_N$ the frequencies of each distinct address, the entropy of X is given by:

$$H(X) = \log_2(m) - \frac{1}{m}\sum_{x=0}^{N} f_x \log_2(f_x), \qquad (1)$$

where the summation $S = \sum_{x=0}^{N} f_x \log_2(f_x)$ is the entropy norm. Note that the entropy norm has a negative relation to the entropy itself. The minimum entropy $H = 0$ occurs when all addresses are the same such that $S = m\log_2(m)$. Dispersed distributions result in higher entropy values reaching the maximum $H = \log_2(m)$ when all addresses are distinct, i.e., $S = 0$.

In the course of a DDoS attack, we expect the entropy of source IP addresses to increase as the malicious packets introduce new values to the distribution. Conversely, we expect the entropy of destination IP addresses to decrease with the victim becoming more frequent as a destination. This effect is only observable when the number of packets m encompasses an adequate, robust representation of the current distributions. One must keep in mind that increasing m comes at the cost of higher attack detection latency, as more packets must be received for each measurement. On the other hand, when calculating the entropy over few packets, malicious traffic-related changes to the distributions may be indistinguishable from short-term fluctuations of legitimate traffic.

To address the mentioned trade-off, we propose setting dynamic thresholds to the entropies of the source and the destination IP addresses considering a preset value of m. In the following subsection, we present the packet processing pipeline devised to implement this approach.

C. Packet Processing Pipeline

We build the detection mechanism on top of the P4 behavioral model reference implementation (BMv2) [25], which has a constrained set of processing primitives reflecting the limitations of the current programmable hardware devices. In this subsection, we describe how we overcome these restrictions to perform real-time DDoS attack detection.

Figure 1 depicts the top-level scheme of our proposed mechanism. The entropies of IP addresses are estimated for consecutive partitions of the incoming packet stream, named *observation windows* (Section III-C1). At the end of each observation window, the *traffic characterization* units read the entropy values to generate a legitimate traffic model (Section III-C2). In turn, the *anomaly detection* unit calculates *detection thresholds* as functions of this model issuing an *attack alarm* when they are exceeded by the last entropy estimates (Section III-C3).

1) Entropy Estimation: As the P4 behavioral model does not support the binary logarithm function, we assume a fixed value (yet parameterizable) for the observation window size

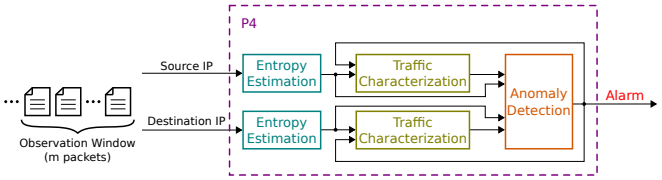

Figure 1. DDoS Attack Detection Top-Level Scheme

m so that the first term in Equation 1 becomes a constant. Consequently, the real-time entropy estimation processing is reduced to the calculation of the second term, which is a function of the frequencies of each distinct observed address.

Next, we detail the approximation of these frequencies from the packet stream while meeting processing constraints. Then, we elaborate on the computation of the entropy norm without resorting to an offline processing stage. Finally, we show how to obtain the entropy estimate. Figure 2 illustrates the processing pipeline in its entirety.

a) Frequency approximation: We base the approximation of address frequencies on a *count sketch* data structure [14], which uses sub-linear space to represent a frequency table of events in a data stream. It requires the computation of hash functions and a two-dimensional array of counters to obtain unbiased probabilistic frequency approximations.

Let X be the set of all possible IP address values and C be a matrix of counters with depth d and width w (i.e., $C \in \mathbb{Z}^{d \times w}$), where $C_{i,j}$ indicates the counter at row i and column j (see Figure 2). We define two sets of independent hash functions $\{h_1, ..., h_d\}$ and $\{g_1, ..., g_d\}$, where each pair (h_i, g_i) is associated with a sketch row $i \in \{1, ..., d\}$. All hash functions have as input an IP address $x \in X$. Hash function h_i maps IP addresses to columns in row i (i.e., $h_i : X \mapsto \{1, ..., w\}$). Hash function g_i decides, for each IP address, if the counter $C_{i,h_i(x)}$ should be incremented or decremented (i.e., $g_i : X \mapsto \{-1, 1\}$). The count sketch algorithm defines two operations:

$Update(C, x)$:
  for $i = 1, ..., d$:
    $C_{i,h_i(x)} \leftarrow C_{i,h_i(x)} + g_i(x)$
$Estimate(C, x)$:
  return $median(g_1(x)C_{1,h_1(x)}, ..., g_d(x)C_{d,h_d(x)})$

$Update(C, x)$ updates all depth levels of the sketch C to count the occurrence of x. $Estimate(C, x)$ returns an estimate of the frequency count of x, which we denote as $\hat{f}_x$ (see Figure 2). The mechanism uses the set of hash functions g_i to deal with the event of h_i colliding for multiple distinct IP addresses. In that event, it is expected that some of the addresses will increase the counter value and others will decrease it, making the counter assume a clearly inconsistent value. When compared with the other counters of the same address stored on all depth levels, counters with collisions become outliers. The sketch avoids getting tainted by such outliers by using the median (instead of the mean, which is very sensitive to outliers) of the values stored in all depth levels as the frequency estimate.

In P4, the mentioned data structure can be implemented using registers, which persist general purpose data across packets. IP address hashing is possible through the definition of custom hash functions. We use functions of the type $(a_i x + b_i) \bmod p$, where a_i and b_i are co-prime coefficients, and p is a prime number. These functions have been successfully used before in programmable data planes [26].

Obtaining independent consecutive entropy estimates would require to reset all sketch counters at the transition of observation windows. To avoid such a bursty processing overhead, we associate an additional register to each sketch counter to store the identifier of the observation window in which it was last updated (W_{ID}). We employ an observation window counter to generate these identifiers. Hence, whenever a counter is read, its value is only taken into account if the associated register holds the current window identifier; otherwise, it is presumed zero and updated accordingly.

Finally, we take the median value comparing the results of each sketch row. Note that the sketch depth is equal to the number of inputs to the median operator. Thus, such parameter is intrinsically related to the processing complexity of this step.

b) Entropy norm estimation: Right after an IP address is read, and its current frequency on the observation window is retrieved, we compute its respective term in the summation composing S. We use this result to update the entropy norm estimate $\hat{S}$ (stored in a register). As IP addresses are expected to appear numerous times in a single observation window, we update $\hat{S}$ by incrementing the difference between the newly-computed term and its previous value (if $\hat{f}_x > 1$), as follows:

$$\hat{S} \leftarrow \hat{S} + \underbrace{\hat{f}_x \log_2(\hat{f}_x)}_{\text{newly-computed term}} - \underbrace{(\hat{f}_x - 1)\log_2(\hat{f}_x - 1)}_{\text{previous term value}}. \quad (2)$$

Since the P4 behavioral model does not support floating-point numbers, we represent $\hat{S}$ in a fixed-point format to allow fractional precision. The required arithmetic operations can be derived from integers.

As an important building block for implementing Equation 2 in the data plane, we must compute the binary logarithm, which is not as straightforward. To overcome this challenge, we build an *LPM lookup table* with pre-calculated values for: $\hat{f}_x \log_2(\hat{f}_x) - (\hat{f}_x - 1)\log_2(\hat{f}_x - 1)$. Unlike a typical lookup table requiring an entry for each domain value, longest-prefix matching allows the aggregation of domain values to a single entry. This data structure is typically supported in a forwarding device by a Ternary Content-Addressable Memory (TCAM). Thus, we replace real-time compute-intensive operations with efficient TCAM table lookups.

Our pre-computed function is plotted in Figure 3. The dashed lines illustrate the aggregation of the domain values $[147\,456, 155\,647]$ to a single entry with the result set to $y = 18.65214$. In this case, the maximum approximation error is $\approx$0.04 when $f_x = 147\,456$. In general, the magnitude of the error depends on the function curve within the aggregation interval. One should wisely populate the table to meet an adequate trade-off between table entry count and error.

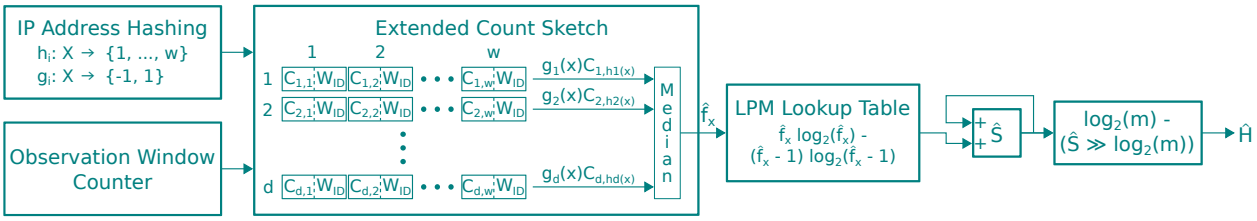

Figure 2. Entropy Estimation Pipeline

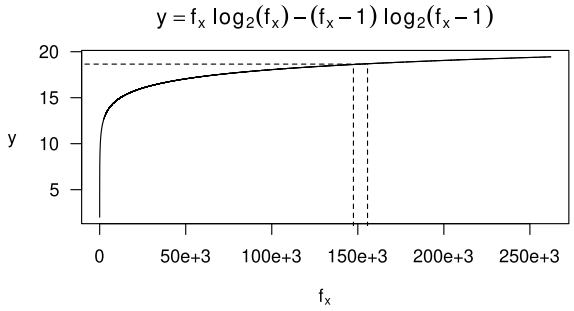

Figure 3. LPM lookup table pre-computed function: the dashed lines illustrate how f_x values can be aggregated to a single table entry with reduced approximation error.

Throughout processing, every incoming packet x triggers $Update(C, x)$ and $\hat{f}_x \leftarrow Estimate(C, x)$. Then, the values of $\hat{f}_x$ are used as keys to obtain the increment to the entropy norm (Equation 2). By the end of each observation window, we calculate the entropy estimates from $\hat{S}$ as shown below.

c) Entropy measurement: In the interest of reducing the processing requirements, we constrain the observation window size to a fixed power of two so that $\log_2(m)$ results in an integer constant and S/m can be implemented as a simple arithmetic shift. Therefore, the entropy estimate is given by:

$$\hat{H} \leftarrow \log_2(m) - (\hat{S} \gg \log_2(m)), \tag{3}$$

where $\gg$ denotes an arithmetic shift. We store $\log_2(m)$ within a register so the network operator can change m at runtime.

2) Traffic characterization: Anomaly-based intrusion detection has the advantage of dealing with attacks of unknown anatomy and different strengths, but usually requires a bootstrapping, training phase with legitimate traffic. In our proposed mechanism, we also build a model of the legitimate traffic, through the processing of successive entropy estimates.

The entropy time series of the source and the destination IP addresses are summarized independently in terms of an index of central tendency and an index of dispersion. Since the address distributions are legitimately subject to changes throughout time, the proposed mechanism updates this model in real-time. The entropy measurements identified as malicious by the anomaly detection unit are discarded from the characterization.

a) Index of central tendency: We represent the central tendency of the recent entropy estimates by their *Exponentially Weighted Moving Average* (EWMA) [27], as follows:

$$M_n \leftarrow \alpha\hat{H}_n + (1-\alpha)M_{n-1} \quad \text{with} \quad M_1 = \hat{H}_1, \tag{4}$$

where $\alpha \in (0, 1)$ is known as the smoothing coefficient and n is an index representing the observation window. This metric allows parameterizable filtering of short-term fluctuations while giving prominence to the most recent values. In our P4-based design, we make use of a fixed-point representation for the smoothing coefficient.

b) Index of dispersion: Likewise, we measure the dispersion of entropy values through an *Exponentially Weighted Moving Mean Difference* (EWMMD), as follows:

$$D_n \leftarrow \alpha|M_n - \hat{H}_n| + (1-\alpha)D_{n-1} \quad \text{with} \quad D_1 = 0. \tag{5}$$

This index denotes the typical spread of entropy measurements relative to the EWMA, being a fundamental feature to the definition of anomaly detection thresholds.

3) Anomaly Detection: The anomaly detection is performed according to the following conditions:

$$\textit{source IP addresses:} \quad \hat{H}_n > M_{n-1} + kD_{n-1} \tag{6}$$

$$\textit{destination IP addresses:} \quad \hat{H}_n < M_{n-1} - kD_{n-1} \tag{7}$$

A DDoS attack alarm is triggered whenever any of these two conditions hold. *k* is a configurable parameter proposed as a *sensitivity coefficient*, which scales the detection thresholds. Since it is multiplied by the index of dispersion, this effect is proportional to the traffic characteristics. Increasing *k* results in more rigorous detection conditions, i.e., higher assertiveness. "Stealthier" attacks may go unnoticed, nevertheless. A lower value for *k*, in turn, may expand anomaly detection coverage with the cost of escalating false alarms. It is up to the network operators to determine and adaptively change a value for *k* to reach an adequate balance between true-positive and false-positive rates.

IV. Evaluation

As far as we are aware of, this work is the first to explore data plane programmability, more specifically P4, to devise a sophisticated anomaly detection mechanism. Given the limited primitive set made available by P4 and the consequent simplifications that were mandatory in our design, it is paramount to assess the *accuracy*, *resource utilization* and *timeliness* of our proposed mechanism, comparing it with state-of-the-art approaches. In this section, we evaluate it aiming to answer the following three research questions:

- *RQ1: How accurate is the entropy estimation processing pipeline as a function of resource utilization footprint?*
- *RQ2: Assuming decent entropy estimation (RQ1), how accurate is the DDoS attack detection mechanism under different tuning parameters and attack strengths?*
- *RQ3: How does our mechanism compare to existing monitoring approaches regarding detection accuracy and latency?*

First, in Section IV-A, we describe the experimental setup and the evaluation methodology. Then, in each of the remaining subsections, we discuss one of the questions above.

A. Experimental Setup and Evaluation Methodology

Given the novelty of P4 and the still scarce availability of equipment implementing it, we evaluate the proposed DDoS attack detection mechanism using a software-based P4 infrastructure. This setup does not limit our assessment, as both accuracy and resource utilization are expected to be equivalent regardless of the P4 target.

We use as legitimate traffic packet traces from the *CAIDA Anonymized Internet Traces 2016* [28] dataset, recorded from high-speed Internet backbone links. To represent DDoS attacks, we take packets from the *CAIDA DDoS Attack 2007* dataset, consisting of an attempt to consume the computing resources of a target server and to congest the network links connecting this server to the Internet. Despite not recent, this dataset was carefully built to only include attack-related traces and, for this reason, is consistently employed in high-impact publications in the area of network security. Furthermore, this choice is consistent with the scenario we introduced earlier in Section III-A.

We set the observation window size m to 2^{18}, representing approximately $250\,\mathrm{ms}$ for the given workload average packet rate. We partition the workload into a training and a detection phase with respectively 250 and 500 observation windows. The training phase consists of only legitimate traffic serving the purpose of setting up the characterization model. For the detection phase, we take the subsequent legitimate traffic and superimpose it with attack packets from the 126th to the 375th observation window. We perform such superimposition at different malicious-to-legitimate packets proportions (3%, 3.5%, ..., 6%), generating a total of 7 workloads.

Table I summarizes the system factor levels set throughout the experiments. We select varying ranges for data structure size and sensitivity coefficient to allow a broad assessment of the proposed mechanism. We execute 15 repetitions for each configuration with random hashing coefficients and present the results using a 95% confidence level. In Section IV-B, we examine the relative error of the entropy estimates for different count sketch depth and width. In Section IV-C, we investigate the detection True-Positive Rate (TPR), False-Positive Rate (FPR), and accuracy with respect to the sensitivity coefficient, the memory footprint, and the different proportions of malicious traffic volume. Finally, in Section IV-D, we compare our mechanism with approaches based on packet sampling regarding detection accuracy and latency.

B. Entropy Estimation Error

Instead of focusing on the calculation of exact entropy values, we propose an estimation processing pipeline that minimizes memory space and processing time. However, the loss of accuracy in this process has the potential to undermine the detection performance by hiding anomalous traffic patterns. On that account, we assess the relative error of the estimates as a function of the count sketch dimensions, which represent the dominant influential factors to correctness (RQ1).

TABLE I
SYSTEM FACTOR LEVELS

System Factors	Levels Used in Each Subsection		
	IV-B	IV-C	IV-D
Hashing Coefficients (a_i, b_i)	random		
Count Sketch Depth (d)	$\{4, 8, 16\}$	4	4
Count Sketch Width (w)	$\{64, 368, 672, 976, 1280\}$		1280
Sensitivity Coefficient (k)	NA	$\{0, 0.5, 1, ..., 8\}$	4

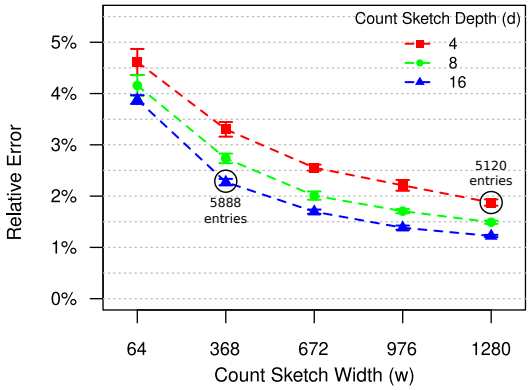

Figure 4. Relative error of the entropy estimation as a function of the count sketch width and depth.

We allocate a 32-bit register for each sketch counter and an 8-bit register for its associated observation window identifier. We store $\hat{S}$ and $\hat{H}$ in 32-bit registers considering a fixed-point representation having 4 fractional bits. We build the LPM lookup table ensuring a maximum error of 2^{-4} for each entry, resulting in a total of 245 TCAM entries.

Figure 4 presents the relative estimation error for the count sketch depth and width levels listed in Table I (first column). The sketch width is directly related to the probability of hashing collisions on each row. Along the horizontal axis, it is possible to observe how this parameter affects the estimation error. The errors reduce as we increase the width, but this reduction attenuates for larger widths and stabilizes close to 1%. Note that this error also results from the approximations present in the pre-computed LPM lookup table entries.

The increment of the sketch depth reduces the probability of getting the estimate from a counter that has been affected by collisions. We observe this effect examining the different error values for a single sketch width. However, increasing the sketch depth implies (*i*) processing more hash functions for each IP address and (*ii*) increasing the complexity of the median operation. The annotations of Figure 4, indicating the total number of sketch entries (5 888 and 5 120) in two specific configurations ($d = 16$ and $d = 4$, respectively), reveal that, for comparably sized sketches, the use of more hashes (rows) does not result in significantly better estimates. Thus, we choose to set $d = 4$ in the subsequent experiments.

C. DDoS Attack Detection Performance

The proposed mechanism allows network operators to adjust the trade-off between the TPR and the FPR using the sensitiv-

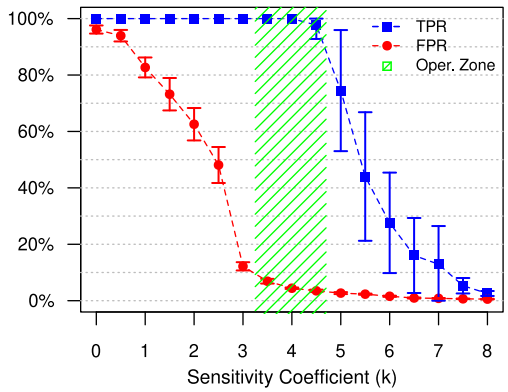

Figure 5. Impact of the sensitivity coefficient k on the true-positive and false-negative rates. The area in green highlights the desired operating zone.

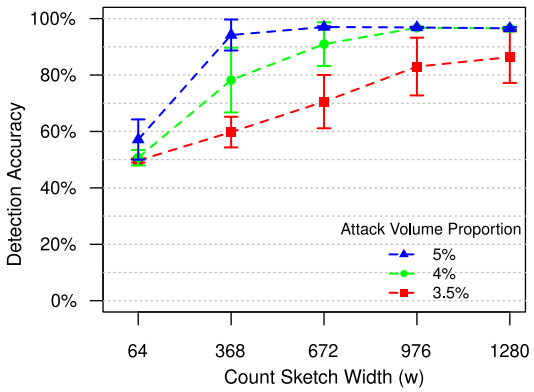

Figure 6. DDoS Attack Detection Accuracy in terms of Memory Utilization for Different Proportions of Malicious Traffic

ity coefficient k. To answer RQ2, we first take these metrics into account to tune this parameter (Section IV-C1). Then, we investigate the detection accuracy regarding malicious traffic volume and memory utilization (Section IV-C2).

1) Sensitivity Coefficient Effect: Figure 5 presents the true-positive and false-negative detection rates in terms of the sensitivity coefficient. The results are for the sketch dimensions $d = 4$, $w = 1\,280$, and the smoothing coefficient $\alpha = 20 \cdot 2^{-8}$. The proportion of the malicious traffic to the overall traffic during the attack is 5%.

Lower sensitivity coefficient values tighten the detection thresholds resulting in higher detection ratios at the cost of false positives. As we increase the coefficient, both the TPR and the FPR decrease to the point where the detection is utterly insensitive. The FPR starts to decrease from $k = 0$ and reaches less than 10% for k within $[3.25, 4.75]$, while the TPR is still close to 100%. This region (green hachure) represents the configuration in which the detection thresholds are expected to be set, i.e., between the entropy estimates of legitimate and malicious traffic. It characterizes the desired operating zone. Given the dynamic nature of traffic in production networks, the value of k may need to be adapted on a periodic basis. This will be addressed in future work.

2) DDoS Attack Detection Accuracy: With the sensitivity coefficient k set to 3.5, we now consider the resulting attack detection accuracy achieved with our proposed mechanism (see Figure 6). The analysis is carried out considering various attack proportions and count sketch widths (see Table I).

As malicious traffic becomes more aggressive, i.e., assumes a higher proportion when compared to the legitimate traffic, the detection accuracy achieves increasingly higher rates (exceeding 90%). This accuracy is profoundly influenced by the count sketch width. Note that even lower magnitude attacks (3.5%) can be decently detected (with rates higher than 80%) as one parameterizes the mechanism with larger w (greater than 976). However, one must recognize that for the cases of lower volume attacks, the anomalous variation of entropy is attenuated and consequently harder to detect. This difficulty is intrinsic to anomaly-based attack detection and is exacerbated when considering lower count sketch widths, which result in less accurate entropy estimates.

If on the one hand, the use of larger sketches leads to higher attack detection accuracy, on the other, it implies a larger memory footprint. Considering 32 bits are allocated for each sketch counter and 8 bits for its associated observation window identifier, the cost for the source and destination IP addresses sketches is $38.125\,\mathrm{kB}$ when $d = 4$ and $w = 976$. This value is the memory space required for monitoring a single $1\,\mathrm{Gbps}$ link. For higher traffic rates, we would need to increase the observation window size to get a robust representation of the addresses distribution. Since the count sketch estimation error is proportional to $1/\sqrt{w}$ and to the square root of the observation ℓ_2 norm [14], we would have to use proportionally larger sketches to obtain an equivalent entropy estimation accuracy. Hence, considering a 24x10 Gbps programmable forwarding device [10], we extrapolate our mechanism memory footprint to $9\,\mathrm{MB}$[2], which represents 18% of the available SRAM.

D. Comparison with Packet Sampling

By collecting information from every packet, programmable forwarding devices have the potential to detect very subtle traffic anomalies. In contrast, packet sampling approaches provide information at a coarser granularity, thus being less sensitive to such conditions. We investigate this difference by comparing our mechanism with an implementation of the same detection strategy fed by an sFlow collector (RQ3).

We evaluate the sFlow implementation with the sampling rate set to 1:1 000, as it is the suggested for a $1\,\mathrm{Gbps}$ link [7], and to 1:100 aiming to get even more optimized results. In order to analyze our approach and the two sFlow-based scenarios considering a comparable baseline, we set different values for m in each implementation seeking to represent approximately the same time duration. For instance, during the time our proposed mechanism reads m packets, the sFlow collector outputs only about $m/1\,000$ or $m/100$ depending on the chosen sampling rate. Hence, we use such scaled values of m for the sampling implementations to normalize the time frame after which they trigger DDoS attack alarms.

Figure 7 depicts the DDoS attack detection accuracy for each approach considering different volume proportions for the malicious traffic. With the sampling rate at 1:1 000, the

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

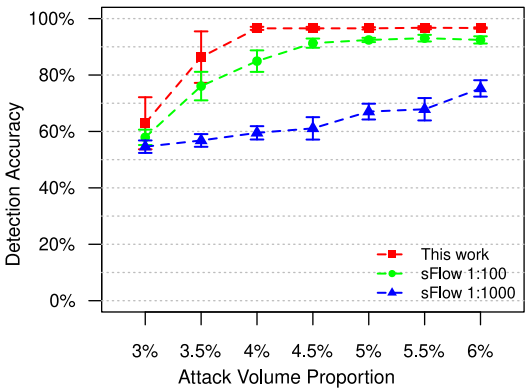

Figure 7. DDoS attack detection accuracy: comparison with packet sampling approaches.

detection performance is severely degraded. At a 1:100 rate, the sFlow approach results are greatly improved, but our work still outperforms its accuracy in every observed condition.

We also consider the detection latency by measuring the time between the timestamp of the first malicious packet and the timestamp of the last packet of the observation window which set off the alarm. For the case of low-intensity attacks ($\leq$4%), we observe increased latency – in the order of seconds – to detect an attack when using packet sampling. Our proposed mechanism for a similar condition requires a fraction of the time (a few hundred miliseconds). The higher sensibility of our proposal may lead to earlier triggering of mitigation actions, possibly preventing service outages.

V. Lessons Learned and Insights

The instruction set available in P4 is very restricted. For example, there is no support for iteration/recursion (except for header parsing), floating-point arithmetic, and non-elementary mathematical functions. These language limitations are due to constraints in the current programmable hardware and help to prevent stalls in the processing pipeline. As a consequence, implementing sophisticated network functions (e.g., anomaly detection, load balancing) in P4 may be challenging. Next, we discuss the major lessons learned in the process of designing the proposed mechanism.

1) Iterative procedures need to be carefully decomposed into small tractable steps triggered by incoming packets: In our work, this observation came from two design challenges: (*i*) the summation of individual address frequencies for entropy estimation (Equation 1) and (*ii*) resetting the sketch counters between observation windows to avoid using outdated values. Iterating over the entire sketch during the processing of a single packet would violate line rate packet processing requirements. Our mechanism handles the challenge (*i*) by calculating the entropy gradually; it only accesses entries relative to the addresses of each incoming packet. We deal with the challenge (*ii*) by augmenting entries with observation window identifiers. A counter value is only used when its identifier is current and its value is only reset when it needs to be updated.

2) Non-elementary mathematical functions may be approximated using LPM lookup tables: Mathematical functions that build upon functions such as logarithm cannot be directly implemented in current programmable devices. When adapting one of these functions to run at the data plane, it is important to analyze its image and argument bounds. Our mechanism uses LPM lookup tables with pre-calculated values to approximate the function for updating the entropy estimate (Equation 2). This function has well-defined argument bounds (i.e., each frequency cannot be higher than the observation window size) and a strict image set (Figure 3). Both of these properties enable having a memory-efficient LPM table by compressing entries with close values without significant loss in accuracy.

3) Floating-point support may not be essential to express numbers with a specific precision in a confined known range: Traditional packet forwarding does not require floating-point arithmetic. Thus, forwarding devices typically only provide instructions over integers. As an upside, integer arithmetic can be applied to handle fractional numbers assuming a fixed-point representation. Throughout our work, we use a fixed-point representation with a 2^4 scaling factor to express real numbers. These numbers in our mechanism are the entropy norm, the entropy itself, the smoothing coefficient, and both the indices of central tendency and of dispersion. Our evaluation shows that it is sufficiently accurate to detect DDoS attacks.

4) The absence of dynamic memory allocation functionality in the data plane can hamper mechanism self-tuning: The proposed mechanism has some parameters (i.e., m, k, and α) that can be modified at runtime through register updates. Other parameters (i.e, d and w) cannot be changed by the in-switch logic, demanding a new P4 program to be deployed on the forwarding devices by an external controller. The reason is that a P4 program cannot allocate dynamic memory. The implementation of more complex self-tuning capability, therefore, requires the investigation of novel data structures.

VI. Conclusion

In this paper, we presented a real-time DDoS attack detection mechanism, implemented with P4, to be executed entirely in the data plane. This work underscores the potential of our P4-based design towards meeting increasingly strict monitoring requirements. The evaluation results show that the mechanism can detect DDoS outbreaks quickly and accurately, especially when compared with existing monitoring approaches, while meeting strict memory space and processing time budgets associated with in-network packet processing. As another significant contribution, we shared lessons learned with our design and implementation, expecting they are valuable for new developments in the area. In future work, we intend to explore further the possibilities of P4 by proposing an Artificial Intelligence-based Anomaly Detection unit and providing the mechanism with self-tuning capability.

Acknowledgements

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), the Brazilian National Research and Educational Network (RNP), and the National Science Foundation (NSF).

REFERENCES

[1] D. Anstee, C. F. Chui, P. Bowen, and G. Sockrider, "Worldwide infrastructure security report," Arbor Networks, Tech. Rep. XII, 2017.

[2] S. Hilton. (2016, October) Dyn analysis summary of friday october 21 attack. Oracle Dyn. [Online]. Available: https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/

[3] S. Kottler. (2018, March) February 28th DDoS incident report. GitHub. [Online]. Available: https://githubengineering.com/ddos-incident-report/

[4] P. Phaal, S. Panchen, and N. McKee, "InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks," Internet Requests for Comments, RFC Editor, RFC 3176, September 2001. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3176.txt

[5] B. Claise, "Cisco Systems NetFlow Services Export Version 9," Internet Requests for Comments, RFC Editor, RFC 3954, October 2004. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3954.txt

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

[7] P. Phaal. (2009, June) sflow: Sampling rates. [Online]. Available: https://blog.sflow.com/2009/06/sampling-rates.html

[8] M. Moshref, M. Yu, and R. Govindan, "Resource/accuracy tradeoffs in software-defined measurement," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 73–78. [Online]. Available: http://doi.acm.org/10.1145/2491185.2491196

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2656877.2656890

[10] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486011

[11] A. Gupta, R. Birkner, M. Canini, N. Feamster, C. Mac-Stoker, and W. Willinger, "Network monitoring as a streaming analytics problem," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: ACM, 2016, pp. 106–112. [Online]. Available: http://doi.acm.org/10.1145/3005745.3005748

[12] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 85–98. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098829

[13] J. Boite, P. A. Nardin, F. Rebecchi, M. Bouet, and V. Conan, "Statesec: Stateful monitoring for DDoS protection in software defined networks," in *2017 IEEE Conference on Network Softwarization (NetSoft)*, July 2017, pp. 1–9.

[14] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ser. ICALP '02. London, UK, UK: Springer-Verlag, 2002, pp. 693–703. [Online]. Available: http://dl.acm.org/citation.cfm?id=646255.684566

[15] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 2046–2069, Fourth 2013.

[16] Y. Xu and Y. Liu, "DDoS attack detection under SDN context," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

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

[18] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. Lombard, IL: USENIX, 2013, pp. 29–42. [Online]. Available: https://www.usenix.org/conference/nsdi13/technical-sessions/presentation/yu

[19] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16. New York, NY, USA: ACM, 2016, pp. 101–114. [Online]. Available: http://doi.acm.org/10.1145/2934872.2934906

[20] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: ACM, 2018, pp. 561–575. [Online]. Available: http://doi.acm.org/10.1145/3230543.3230544

[21] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in DDoS attacks: Trends and challenges," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2242–2270, Fourthquarter 2015.

[22] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, vol. 27, pp. 623–656, 1948.

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

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

[25] BMv2. P4 Language Consortium. [Online]. Available: https://github.com/p4lang/behavioral-model

[26] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research*, ser. SOSR '17. New York, NY, USA: ACM, 2017, pp. 164–176. [Online]. Available: http://doi.acm.org/10.1145/3050220.3063772

[27] 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

[28] The CAIDA UCSD Anonymized Internet Traces 2016. [Online]. Available: http://www.caida.org/data/passive/passive_2016_dataset.xml