This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

In-Network Volumetric DDoS Victim Identification Using Programmable Commodity Switches

Damu Ding, *Student Member, IEEE*, Marco Savi, Federico Pederzolli, Mauro Campanella, and Domenico Siracusa

Abstract

Volumetric distributed Denial-of-Service (DDoS) attacks have become one of the most significant threats to modern telecommunication networks. However, most existing defense systems require that detection software operates from a centralized monitoring collector, leading to increased traffic load and delayed response. The recent advent of Data Plane Programmability (DPP) enables an alternative solution: threshold-based volumetric DDoS detection can be performed directly in programmable switches to skim only potentially hazardous traffic, to be analyzed in depth at the controller. In this paper, we first introduce the BACON data structure based on sketches, to estimate per-destination flow cardinality, and theoretically analyze it. Then we employ it in a simple in-network DDoS victim identification strategy, INDDoS, to detect the destination IPs for which the number of incoming connections exceeds a pre-defined threshold. We describe its hardware implementation on a Tofino-based programmable switch using the domain-specific P4 language, proving that some limitations imposed by real hardware to safeguard processing speed can be overcome to implement relatively complex packet manipulations. Finally, we present some experimental performance measurements, showing that our programmable switch is able to keep processing packets at line-rate while performing volumetric DDoS detection, and also achieves a high F1 score on DDoS victim identification.

Index Terms

Anomaly detection, Programmable data planes, DDoS victim identification, P4

I. Introduction

Distributed Denial-of-Service (DDoS) attacks are a critical security threat to modern telecommunication networks; not only do they cripple live services for legitimate users, but also cause large operational burdens on operators, which must dedicate significant resources to detecting and mitigating them. As the number and size of botnets and DDoS attacks persistently increases [1], so does this workload. In particular, *volumetric* DDoS attacks, designed to overwhelm network and server capacity, are among the most common and dangerous DDoS attacks [2].

Many techniques to both perform and detect volumetric DDoS attacks are documented in the scientific literature. Among the latter, a rather common feature of such attacks is the exploitation of a (large) number of (capacious) source hosts to direct a considerable amount of packets to a specific victim destination [3][4][5]. Attackers use seemingly legitimate TCP, UDP, or ICMP packets in volumes large enough to overwhelm network devices and servers, or deliberately incomplete packets designed to rapidly consume all available computing, storage, and transmission resources in servers. A majority of such attacks also make use of spoofed source IP addresses (e.g. DNS and NTP amplification DDoS attacks), that is, they forge the address that supposedly generated the requests to prevent source identification; this implies that tracking attack source IPs is usually ineffective. On the other hand, DDoS victim addresses cannot be spoofed, and identifying the victims is a useful step for network operators to mitigate such attacks.

In legacy networks, including SNMP- [6] and NetFlow-based [7] networks, flow monitoring and anomaly/attack detection are executed on top of a centralized *monitoring collector* or *controller*. Such a collector needs to periodically analyze the sampled flow statistics retrieved from network devices to monitor the network state, which has two well-known drawbacks [8]: *(i.)* significant communication overhead is generated between switches and the centralized monitor/controller due to frequent flow sampling, and *(ii.)* large detection latency, which limits the system efficiency in detecting relatively short attacks lasting one or two minutes. Furthermore, it limits the monitoring system ability to detect features that manifest even in sparsely sampled traffic (which may or may not be a problem in the specific case of volumetric DDoS attack detection, but is nonetheless a limitation). The advent of *data-plane programmable switches* (e.g., Tofino [9]) enables the migration of volumetric DDoS attack detection (and victim identification) to the switches themselves, using domain-specific programming languages like P4 [10]. Despite using a relatively high-level language, programming the behaviour of a physical

Damu Ding is with Fondazione Bruno Kessler, Trento, Italy and University of Bologna, Bologna, Italy.
Marco Savi is with University of Milano-Bicocca, Milano, Italy. The study was mainly done while he was with Fondazione Bruno Kessler, Trento, Italy.
Federico Pederzolli and Domenico Siracusa are with Fondazione Bruno Kessler, Trento, Italy.
Mauro Campanella is with GARR, Roma, Italy.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

switch requires specific care in the optimization of physically limited resources to maintain line-rate processing speeds. The language specifications reflect the hardware boundaries: it features the absence of expensive and unbounded operations (e.g., division and loops), due to the limited number of pipeline stages in the processor [11]. Such constraints place a ceiling on the number of operations executable on each packet to ensure line-speed processing.

This paper describes the implementation and validation of volumetric DDoS victim detection and identification directly in Tofino-based [9] P4-enabled commodity switches. The proposed logic copes well with the aforementioned restrictions of data plane programmability. That is, unlike other recent works on this subject, we actually implemented and tested our technique in a real switch instead of limiting the work to a P4 simulator, such as the Behavioral model [12]. To that end, after describing some background mathematical results and data structures required by our technique in Section II, we propose, in Section III, two main contributions: *BACON Sketch* and *INDDoS*. BACON is a new sketch (a probabilistic data structure) combining a Direct Bitmap [13] and a Count-min Sketch [14], which allows switches to estimate the number of distinct flows (i.e., packets with the same flow key) contacting the same destination host. INDDoS is a simple volumetric DDoS victim identification strategy built on top of BACON Sketch to identify the destination IPs contacted by a number of source IPs greater than a threshold, in a given time interval, completely in the data plane. We include extensive theoretical analysis and detail the modifications required to implement our approach in physical resource-constrained P4 Tofino switches in Section IV. Then, Section V provides insights on the integration of INDDoS in a full DDoS defense system, including attack detection and consequent mitigation steps. Section VI presents the results of an evaluation of INDDoS in a small testbed. We show that, using optimal parameters derived from our theoretical analysis, our implementation can reach an F1 score higher than 0.95 on a real flow trace [15] captured on a 10 Gbps backbone link, without performance degradation in the switch's packet-processing capabilities. Finally, Section VII relates our paper to other recent works on the subject while Section VIII concludes it. The work has been carried out within the GÉANT [16] GN4-3 project.

II. Background

A. Markov's inequality

Given a non-negative random variable X and a positive value a, Markov's inequality defines a constant upper bound for the probability that satisfies $\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[X]}{a}$, where $\mathbb{E}[X]$ represents the expected value of X.

B. Direct Bitmap

Direct Bitmap [13] is a simple data structure that can be used to estimate the number of distinct flows occurring in a packet stream (also called *flow cardinality*): it is based on a bit array called *Bitmap register* and relies on one or more *hash functions*. Initially, all m cells in the Bitmap register are set to 0. When a packet arrives, its flow key is hashed: the hashed key indicates the index of the register cell to consider. The value of the indexed cell is set to 1 if it was previously 0, otherwise it is not updated. Note that packets sharing the same flow key are always hashed into the same cell, while different flows are hashed to different cells unless a collision occurs. The number of distinct flows can then be (under-)estimated by counting the number of bits with value 1 in the register.

C. Count-min Sketch

The estimation of the *per-flow packet count* (i.e., number of packets carried by any network flow during an observation interval) can be performed using a *Count-min Sketch* [14], a probabilistic and memory-efficient data structure which implements *Update* and *Query* operations: the former is responsible for continuously updating the sketch to count incoming packets in the switch, while the latter retrieves the estimated number of incoming packets for a specific flow. Count-min Sketch relies on d pairwise-independent hash functions, each with an output of size w. The data structure is composed by a matrix of $d \cdot w$ counters: the accuracy of packet count estimation in Count-min Sketch increases as d or w increase, and vice versa.

III. In-network DDoS victim identification

A. Threat model and deployment scenario

1) Threat model: In this paper, we focus on volumetric DDoS attacks against victim destinations in the network. For the purpose of overwhelming the available resources of the victims, an attacker exploits a large number of distributed hosts (e.g. bots in a botnet) to frequently send traffic to the target host(s) (e.g., a web server). The attacker sources are usually spoofed to evade detection, which, coupled with the fact that each source may send only a small amount of traffic to the victim (in a stealthy volumetric DDoS attack), makes identifying attack sources (also called *superspreaders*) less efficient than focusing on destinations for DDoS mitigation. Unfortunately, there exists legitimate network events, such as flash crowds, that exhibit

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

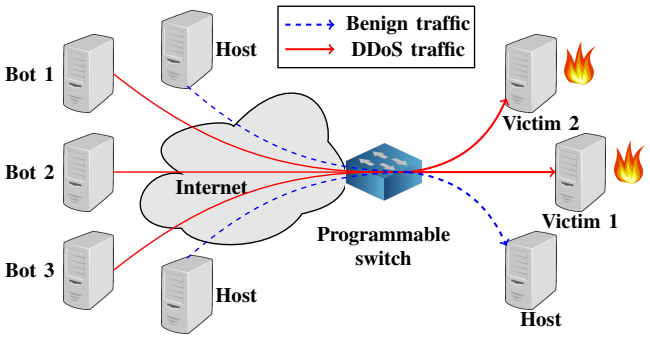


Fig. 1. Deployment scenario

similar characteristics to volumetric DDoS attacks, making a suspiciously high number of sources contacting a destination a necessary but not sufficient condition to determine whether a destination is under attack. To further discriminate DDoS attacks from flash crowds, a possible solution can be found in [17], where the correlation of flows is used to determine whether they are malicious or not.

2) Deployment scenario for the proposed DDoS detection: We target an ISP network, for which the best placement of our DDoS detection functionality involves deploying programmable switches at the edge of the network, so that at least one switch has visibility on all flows towards each IP destination (Fig. 1). Therefore, at least one switch is in the best place to estimate the number of source hosts contacting any destination host. Once a DDoS victim is identified, ideally after a more thorough analysis step at a centralized controller, any border programmable switch can also be used to limit the traffic rate towards it.

B. BACON Sketch

For the purpose of measuring flow cardinality directly inside switches, we combined Direct Bitmap registers [13] and Count-min Sketch [14] in a new sketch, which we named BACON (BitmAp COuNt-min) Sketch for the sake of brevity. As shown in Fig. 2, in BACON Sketch the counters of Count-min Sketch are replaced with a m-sized Bitmap register, hence the size of BACON is $d \times w \times m$, where $d \times w$ is the size of Count-min Sketch. BACON Sketch enables the estimation of per-destination flow cardinality, with flows identified by different flow keys key_{src}, using very little memory in the switch. Two different flow keys are considered for each packet: the key key_{src} must include the source IP of the flow together with any subset of {source port, destination IP and port, protocol} without loss of generality, and the choice of key_{src} depends on the requirements of operators. Likewise, the flow key of the destination host, denoted by key_{dst}, can be either the destination IP or the {destination IP, destination port} pair.

Formally, BACON Sketch solves the following problem. Given:

- a packet stream S
- a Bitmap register size m
- a number of hash functions in Count-min Sketch d
- an output size of hash functions in Count-min Sketch w
- a time interval T_{int}
- a flow key key_{dst}

compute $\hat{E}_{dst}$, the estimated value of E_{dst} in T_{int}, where E_{dst} is the number of flows (identified by different key_{src} keys) that contact the destination host identified by key_{dst}.

Algorithm 1 shows the pseudo code of BACON Sketch. For each incoming packet, the switch hashes the flow key key_{src} with hash function h_{bm} and converts the hashed value to be within range $[0, m-1]$: this value, named $bucket$, is the index in the Bitmap register (Line 8). Then each of the d pairwise-independent hash function in BACON, denoted by h^i_{cm}, hashes the packet's key_{dst} to the slot $(h^i_{cm}(key_{dst})\%w) \cdot m + bucket$ in each row, setting the related register cell to 1 (Lines 9-12). $BACON_i$ refers to the i-th row of BACON Sketch. Note that this way to compute the cell's index to be updated assumes that the Bitmap registers' cells in each row are progressively numbered from 0 to $m \cdot w - 1$, much like in an array. Concerning the *Query* operation for any key_{dst}, the involved Bitmap register in row i (i.e., the register indexes from $(h^i_{cm}(key_{dst})\%w) \cdot m$ to $h^i_{cm}(key_{dst})\%w) \cdot m + m - 1$) estimates the cardinality of key_{dst} by computing the sum of values in the register cells (i.e., number of 1s). This is done for each row and the minimum estimated cardinality $\hat{E}_{dst}$ among all d rows (i.e. $\hat{E}_{dst} = min(E_i)$) is returned (Lines 15-21).

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

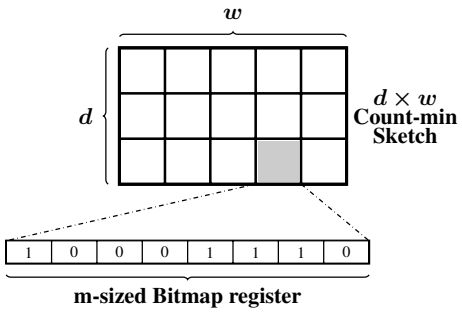


Fig. 2. Data structure of BACON Sketch

Algorithm 1: BACON Sketch

Input: Packet stream S, where each packet is characterized by key_{src} and key_{dst}
Output: Estimated number of distinct flows contacting the packet's destination key_{dst} (i.e., $\hat{E}_{dst}$)

$d \leftarrow$ Number of hash functions h^i_{cm} in Count-min
$w \leftarrow$ Output size of hash functions h^i_{cm} in Count-min
$m \leftarrow$ Bitmap-register size with hash function h_{bm}
for *Each packet in S (with* key_{src} *and* key_{dst}*)* **do**
    Update(key_{src}, key_{dst})
    Query(key_{dst})
Function *Update*(key_{src}, key_{dst})**:**
    $bucket \leftarrow h_{bm}(key_{src})\%m$
    for *Each hash function* h^i_{cm} **do**
        $index \leftarrow (h^i_{cm}(key_{dst})\%w) \cdot m + bucket$
        if $BACON_i[index] == 0$ **then**
            $BACON_i[index] \leftarrow 1$ `// row i`
Function *Query*(key_{dst})**:**
    $\hat{E}_{dst} \leftarrow 0$
    for *Each hash function* h^i_{cm} **do**
        $id \leftarrow (h^i_{cm}(key_{dst})\%w) \cdot m$
        $E_i \leftarrow \sum_{j=id}^{id+m-1} BACON_i[j]$
        if $\hat{E}_{dst} == 0$ **then**
            $\hat{E}_{dst} \leftarrow E_i$
        else if $E_i < \hat{E}_{dst}$ **then**
            $\hat{E}_{dst} \leftarrow E_i$
    return $\hat{E}_{dst}$

C. In-network cardinality-based DDoS victim identification

Using BACON Sketch, we propose a simple in-network volumetric DDoS detection mechanism, which we named INDDoS, aiming to identify likely DDoS victims, i.e., those hosts that are contacted by an abnormally high number of source IPs, and hence the associated network flows.

Formally, the problem is formulated as follows.

Given:

- a number of source IPs n
- a BACON Sketch with size $d \times w \times m$ $(m < n)$
- a DDoS threshold fraction θ (threshold: θn)
- a time interval T_{int}

return all destinations key_{dst}, named *DDoS victims*, that satisfy $\hat{E}_{dst} > \theta n$ within time interval T_{int}, where $\hat{E}_{dst}$ is the

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

estimated number of sources contacting key_{dst} and obtained by querying the BACON Sketch.

From this point onward, without any loss of generality, we will use src (the source IP) as key_{src}, and dst (the destination IP) as key_{dst}, to estimate how many flows from different source IPs are trying to contact a destination IP in a given time interval T_{int}. Observe that the proposed strategy, which requires updating and querying the BACON Sketch for each incoming packet towards any destination *dst* during the time interval, interacts with a remote controller only when at least one attack is detected in the same interval, otherwise no communication between controller and switches takes place.

D. Theoretical analysis

In this section we present a theoretical analysis of the accuracy of our proposed DDoS detection strategy.

1) Error bounds for estimated flow cardinality:

Theorem 1. *With probability at least* $1-(\frac{1}{2})^d$*, the cardinality estimation* $\hat{E}_{dst}$ *from BACON Sketch satisfies* $E_{dst} - 2(n - m(1-e^{-\frac{n}{m}})) < \hat{E}_{dst}$.

Proof. Let us start with the Bitmap register. Since a register cell is touched (i.e., set to 1) as soon as it is selected for one incoming element, the probability of any cell X_i to be touched after processing one element (packet) can be expressed as:

$$\mathbb{P}(X_i = 1) = \frac{1}{m}$$

so, the probability of i-th register cell to be untouched is:

$$\mathbb{P}(X_i = 0) = 1 - \mathbb{P}(X_i = 1) = 1 - \frac{1}{m}.$$

After processing E_{dst} different elements (where E_{dst} is the true number of src contacting dst) we then have:

$$\mathbb{P}(X_i = 0) = (1 - \frac{1}{m})^{E_{dst}} = ((1 - \frac{1}{m})^m)^{\frac{E_{dst}}{m}}.$$

Since $\lim_{m\to\infty}(1-\frac{1}{m})^m = e^{-1}$ and in our case m is large (i.e., ≥ 128), $(1-\frac{1}{m})^m \approx e^{-1}$ and we have:

$$\mathbb{P}(X_i = 0) \approx e^{-\frac{E_{dst}}{m}}.$$

Hence, the probability of a cell to be touched after E_{dst} elements is:

$$\mathbb{P}(X_i = 1) = 1 - \mathbb{P}(X_i = 0) = 1 - e^{-\frac{E_{dst}}{m}}.$$

Since there are m cells in each Bitmap register, the expectation of our Bitmap cardinality-estimation (i.e., the expected number of 1s in the register), for destination IP dst, is:

$$\begin{aligned}\mathbb{E}[\hat{E}_{dst}^{Bitmap}] &= \sum_{i=1}^{m} X_i \cdot \mathbb{P}(X_i = 1) + X_i \cdot \mathbb{P}(X_i = 0) \\ &= \sum_{i=1}^{m} (1 \cdot \mathbb{P}(X_i = 1)) + 0 \\ &= \sum_{i=1}^{m} 1 - e^{-\frac{E_{dst}}{m}} = m(1 - e^{-\frac{E_{dst}}{m}}).\end{aligned}$$

Note that, when there are no collisions in Count-min Sketch, $\mathbb{E}[\hat{E}_{dst}] = \mathbb{E}[\hat{E}_{dst}^{Bitmap}]$. Since $\hat{E}_{dst}^{Bitmap} \leq E_{dst}$, the difference between $\mathbb{E}[\hat{E}_{dst}]$ and $\mathbb{E}[E_{dst}]$[1] is:

$$f(E_{dst}) = \mathbb{E}[E_{dst}] - \mathbb{E}[\hat{E}_{dst}] = E_{dst} - m(1 - e^{-\frac{E_{dst}}{m}}).$$

The derivative of $f(E_{dst})$, denoted by $f'(E_{dst})$, is:

$$f'(E_{dst}) = 1 - e^{-\frac{E_{dst}}{m}}.$$

[1]The expectation of E_{dst} is still E_{dst} since E_{dst} is a constant.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

Since $e^{-\frac{E_{dst}}{m}} < 1$, $f'(E_{dst}) > 0$. For this reason, $f(E_{dst})$ is monotonically increasing and $f(E_{dst}) \leq f(n) = n - m(1 - e^{-\frac{n}{m}})$. Then, applying Markov's inequality yields:

$$\begin{aligned}\mathbb{P}(E_{dst} - \hat{E}_{dst} \geq 2(n - m(1 - e^{-\frac{n}{m}}))) &\leq \frac{E_{dst} - \mathbb{E}[\hat{E}_{dst}]}{2(n - m(1 - e^{-\frac{n}{m}}))} \\ &\leq \frac{n - m(1 - e^{-\frac{n}{m}})}{2(n - m(1 - e^{\frac{-n}{m}}))} = \frac{1}{2}.\end{aligned}$$

Considering that there are d hash functions in the Count-min Sketch part of BACON Sketch:

$$\mathbb{P}((E_{dst} - \hat{E}_{dst}) \geq 2(n - m(1 - e^{-\frac{n}{m}}))) \leq (\frac{1}{2})^d$$

Thus, with probability at least $1 - (\frac{1}{2})^d$,

$$\hat{E}_{dst} > E_{dst} - 2(n - m(1 - e^{-\frac{n}{m}})).$$ □

Theorem 2. *With probability at least $1 - (\frac{1}{2})^d$, the cardinality estimation $\hat{E}_{dst}$ from BACON Sketch satisfies $\hat{E}_{dst} \leq E_{dst} + 2m(1 - e^{-\frac{n}{mw}})$.*

Proof. Count-min Sketch occasionally writes two different inputs to the same cell; the expected number of such collisions is $\frac{n - E_{dst}}{w}$ [14]. Therefore, in BACON Sketch, the actual average number of different elements being written to the same Bitmap register is $E_{dst} + \frac{n - E_{dst}}{w} < E_{dst} + \frac{n}{w}$, so the expectation of $\hat{E}_{dst}$ becomes:

$$\begin{aligned}\mathbb{E}[\hat{E}_{dst}] = \sum_{i=1}^{m} \mathbb{E}[X_i] &= \sum_{i=1}^{m} 1 - e^{-\frac{E_{dst} + \frac{n - E_{dst}}{w}}{m}} \\ &= m(1 - e^{-\frac{E_{dst} + \frac{n - E_{dst}}{w}}{m}}) \\ &< m(1 - e^{-\frac{E_{dst} + \frac{n}{w}}{m}}).\end{aligned}$$

Then, the expectation of $\hat{E}_{dst} - \hat{E}_{dst}^{Bitmap}$ is:

$$\begin{aligned}\mathbb{E}[\hat{E}_{dst} - \hat{E}_{dst}^{Bitmap}] &= \mathbb{E}[\hat{E}_{dst}] - \mathbb{E}[\hat{E}_{dst}^{Bitmap}] \\ &< m(1 - e^{-\frac{E_{dst} + \frac{n}{w}}{m}}) - m(1 - e^{-\frac{E_{dst}}{m}}) \\ &= me^{-\frac{E_{dst}}{m}}(1 - e^{-\frac{n}{mw}}) < m(1 - e^{-\frac{n}{mw}}).\end{aligned}$$

According to Markov's inequality,

$$\begin{aligned}\mathbb{P}((\hat{E}_{dst} - \hat{E}_{dst}^{Bitmap}) \geq 2m(1 - e^{-\frac{n}{mw}})) &\leq \frac{\mathbb{E}[\hat{E}_{dst}] - \mathbb{E}[\hat{E}_{dst}^{Bitmap}]}{2m(1 - e^{-\frac{n}{mw}})} \\ &= \frac{m(1 - e^{-\frac{n}{mw}})}{2m(1 - e^{-\frac{n}{mw}})} = \frac{1}{2}.\end{aligned}$$

Since there are d hash functions in Count-min Sketch:

$$\mathbb{P}((\hat{E}_{dst} - \hat{E}_{dst}^{Bitmap}) \geq 2m(1 - e^{-\frac{n}{mw}})) \leq (\frac{1}{2})^d.$$

Hence, with probability at least $1 - (\frac{1}{2})^d$,

$$\hat{E}_{dst} - \hat{E}_{dst}^{Bitmap} < 2m(1 - e^{-\frac{n}{mw}}).$$

Since $\hat{E}_{dst}^{Bitmap} \leq E_{dst}$, $\hat{E}_{dst}$ satisfies:

$$\hat{E}_{dst} < 2m(1 - e^{-\frac{n}{mw}}) + E_{dst}.$$ □

Remark. *The lower error bound for the cardinality estimation is $E_{dst} - 2(n - m(1 - e^{-\frac{n}{m}}))$, while the upper error bound is $E_{dst} + 2m(1 - e^{-\frac{n}{mw}})$. This implies that we should carefully choose m and w instead of setting the sketch width (i.e., number of columns) as large as possible, due to their counteracting effect on estimation quality.*

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

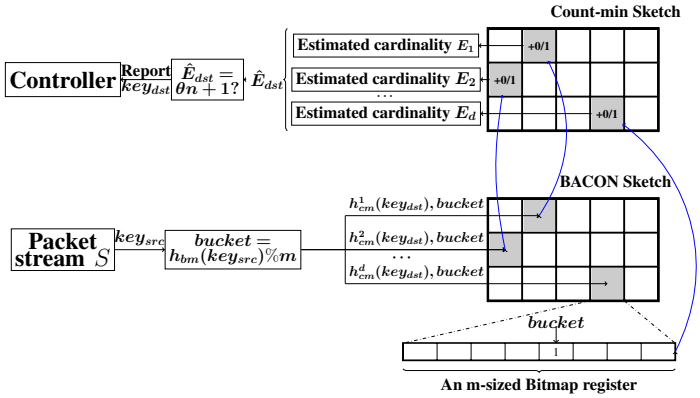


Fig. 3. Scheme of INDDoS including Update (bottom) and Query (top) operations on BACON Sketch as implemented in the commodity switch

2) Error bounds for DDoS detection:

Lemma 1. *With probability at least* $1-(\frac{1}{2})^d$, $R \leq \frac{2n}{w}$, *where* R *is the overestimation of* E_{dst} *caused by Count-min Sketch.*

Proof. The expectation of R in each Bitmap register is $\mathbb{E}[R] = \frac{n-E_{dst}}{w} \leq \frac{n}{w}$, applying Markov's inequality yields:

$$\mathbb{P}(R \geq \frac{2n}{w}) \leq (\frac{1}{2})^d. \qquad \square$$

Given Lemma 1, Theorem 3 reports the *false negative bound* and Theorem 4 the *false positive bound* for INDDoS.

Theorem 3. *When* $E_{dst} \geq \theta n + 2(n - m(1-e^{-\frac{n}{m}}))$, *INDDoS reports dst as a DDoS victim with a probability of at least* $1-(\frac{1}{2})^d$.

Proof. A victim should be reported if $E_{dst} > \theta n$. By Theorem 1, $\hat{E}_{dst} \geq E_{dst} - 2(n - m(1-e^{-\frac{n}{m}})) \geq \theta n$ with probability at least $1-(\frac{1}{2})^d$. Therefore, dst is reported as a DDoS victim with the same probability if and only if $\hat{E}_{dst} \geq E_{dst} - 2(n - m(1-e^{-\frac{n}{m}}))$. $\square$

Theorem 4. *When* $E_{dst} \leq \frac{2}{w}n$ *and* $\theta \geq \frac{4}{w}$, *BACON Sketch reports dst as a victim with a probability of at most* $(\frac{1}{2})^d$.

Proof.

$$\begin{aligned}
\mathbb{P}(\hat{E}_{dst} \geq \theta n) &\leq \mathbb{P}(E_{dst} + R \geq \theta n) \\
&\leq \mathbb{P}(R + \frac{2n}{w} \geq \theta n) \qquad (\textit{due to } E_{dst} \leq \frac{2}{w}n) \\
&= \mathbb{P}(R \geq (\theta - \frac{2}{w})n) \\
&\leq \mathbb{P}(R \geq (\frac{4}{w} - \frac{2}{w})n) \qquad (\textit{due to } \theta \geq \frac{4}{w}) \\
&= \mathbb{P}(R \geq \frac{2}{w}n) \leq (\frac{1}{2})^d \qquad (\textit{by Lemma 1}). \square
\end{aligned}$$

Remark. *The false negative and positive bounds for INDDoS show that increasing the BACON Sketch height (i.e., number of rows) improves the victim identification performance (with equal column parameters).*

IV. Implementation in Commodity Switches

Fig. 3 shows a schematic representation of our in-network DDoS detection implementation within P4-enabled commodity switches. We implemented INDDoS with 680 lines of P4_16 code, released as open source at [18]. Here we report the details of the implementation of BACON Sketch and INDDoS in the Tofino-based hardware switch.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

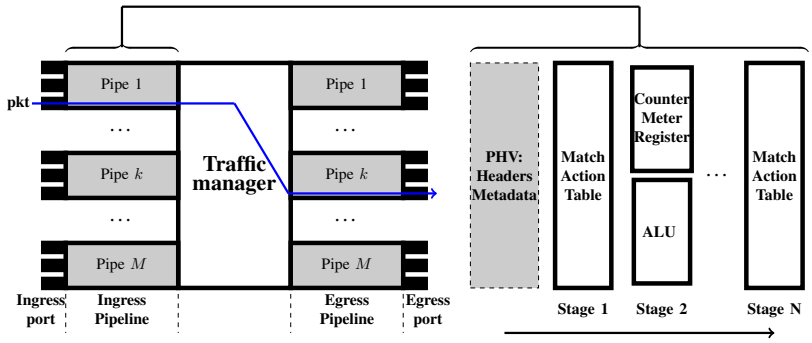


Fig. 4. Tofino-based switch data plane architecture

TABLE I
PROPERTIES OF HASH FUNCTIONS

Hash function name	poly	Reversed	init	xor
CRC32	0x104C11DB7	True	0	0xFFFFFFFF
CRC32c	0x11EDC6F41	True	0	0xFFFFFFFF
CRC32d	0x1A833982B	True	0	0xFFFFFFFF
CRC32q	0x1814141AB	False	0	0
CRC32mpeg	0x104C11DB7	False	0xFFFFFFFF	0

A. *Tofino-based switch data plane architecture*

Fig. 4 shows the architecture of programmable commodity switches with Tofino ASIC (Application-Specific Integrated Circuit). There are multiple *pipes* (each composed by an *ingress* and *egress pipeline*) in the switch, and several ports are associated with one ingress or egress pipeline. Programs executed in different pipes are independent in terms of memory and computational resources. For instance, register values in pipe 1 cannot be read by programs running in pipe 2. When packets enter the switch, they are first processed by the input port's ingress pipeline, then, after crossing the switching matrix, are processed again by the output port's egress pipeline. Each pipeline contains a limited number of *stages*, each including one or more code *blocks* (i.e., sets of operations) that are applied to each packet in sequence. Boundaries on the number of operations executed within each stage exist to ensure that the ASIC can process packets at line rate, irrespective of the custom logic being implemented. Blocks can contain operations including *(i.)* applying match-action tables for customized packet processing, *(ii.)* reading or writing counters/meters/registers for counting or storing packets, and *(iii.)* calls to Arithmetic Logic Units (ALUs) for local computations. Note that each stage has limited hardware resources, such as memory size (Static and Ternary Random Access Memory, i.e., SRAM and TCAM) and number of ALUs. Our implementation of INDDoS fits entirely in the ingress pipeline, leaving the egress pipeline free for other uses as discussed in Section V-C.

B. *Implementation of pairwise independent hash functions*

Since our P4-enabled commodity switch does not natively provide support for pairwise independent hash functions, we varied the usual *poly*, *reverse*, *initial value*, and *xor* parameters in the available embedded CRC32 function to produce a set of suitable pairwise independent hash functions, such as CRC32c, CRC32d, CRC32q and CRC32mpeg (see Table I). These hash functions guarantee that the hashed values of the same flow key are independent.

C. *BACON Sketch - Implementation of updates*

We implemented BACON Sketch with d rows of $w \times m$-sized registers. Considering that we cannot use more than two arithmetic operations to compute the input of a register (due to the atomic action rule [11]), we cannot calculate the $index$ of the cell to be updated as $(h^i_{cm}(dst)\%w) \cdot m + bucket$ (see Alg. 1). Thus, we express $index$ by concatenating $bucket$ and h^i_{cm} as a binary number (see Fig. 5), where the least significant bits $index[0 : \log_2(m) - 1]$ represent $bucket$, while the rest (i.e., $index[\log_2(m) : \log_2(m) + \log_2(w) - 1]$) represent $h^i_{cm}(dst)$. In order to assure that $\log_2(w)$ and $\log_2(m)$ are integers, w and m must be powers of two.

The maximum size of registers in the switch is 2^{17}, which means that if $\log_2(m) + \log_2(w)$ is larger than 17, the concatenated $index$ is truncated to 17 bits. In order to overcome this limitation, it is possible, as shown in Fig. 5, to initialize up to $\log_2(m) + \log_2(w) - 17$ registers in the switch, labeled from 0 to $\log_2(m) + \log_2(w) - 18$. The most significant bits of $index$ are the label of the registers that should be updated. For instance, if $index[17 :] = k$, the index $index[0 : 16]$ at the register labeled k will be updated to 1. Unfortunately, this approach requires few additional pipeline stages in the switch due to using more registers.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

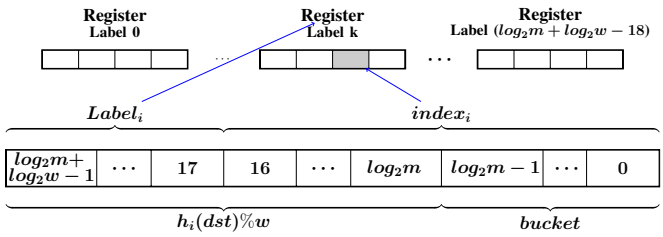


Fig. 5. BACON Sketch updates to overcome the switch's hardware limitations

D. BACON Sketch - Implementation of queries

Calculating the sum of values in a Bitmap register is a costly operation inside a switch, since it needs to iteratively load and compute the sum of many individual values. For this reason, we used an *auxiliary* Count-min Sketch to directly store the updated sum of each Bitmap register (see Fig. 3). This Count-min Sketch is composed of $d \times w$ counters, each associated with the number of 1s in the corresponding Bitmap register of BACON Sketch. This sketch is updated as follows: when a packet arrives, the $h^i_{cm}(dst)$-th counter may *(i.)* stay the same if $index_i$ in row i of BACON Sketch is already 1 or *(ii.)* increase by 1 if the same index of BACON Sketch is 0. The updated value E_i in the counter $index_i$ is the same as the queried value in row i of BACON Sketch. Finally, the flow cardinality of dst is obtained by using *if-else statements* to take the minimum of estimated cardinalities E_i in each row of the auxiliary sketch.

E. Implementation of INDDoS

If a queried flow cardinality $\hat{E}_{dst}$ is equal to $\theta n + 1$ for one or more destinations, the switch's data plane prepares a short *digest* packing the 4 bytes of the detected destination IP(s), and sends it to the controller. The equality prevents generating multiple digests for the same dst in one time interval. At the end of a time interval the switch resets all registers and starts a new round of DDoS victim identification.

F. Limitations hindering the implementation of other solutions

In this subsection, we briefly explain why competing state-of-the-art sketch-based per-flow cardinality estimation algorithms cannot currently be implemented in P4-enabled programmable hardware switches:

1) Virtual HyperLogLog (vHLL) [19]: we were not able to implement this strategy for three reasons: *(i.)* we could not find a way to count tailing 0s for the Update operation in HyperLogLog; *(ii.)* the required harmonic mean for querying cardinality is currently not implementable since P4 does not support the calculation of inverse numbers; *(iii.)* vHLL needs to access several register cells to query the cardinality of a single flow but, in our hardware, the same register cannot be accessed more than once per packet.

2) SpreadSketch [20] (combination of Multiresolution Bitmap [13] and Count-min Sketch [14]): we were not able to implement this strategy for two reasons: *(i.)* P4 does not support the computation of logarithms, which is necessary for Multiresolution Bitmap to estimate the cardinality (we proposed a logarithm estimation algorithm in P4 in [21], but currently it can only be implemented in Behavioral model and not in hardware); *(ii.)* arithmetic operations on the same metadata, in our hardware, cannot occur more than twice, while more are required here.

Thus, unfortunately, we cannot perform a direct comparison of our strategy with the state of the art since the solutions proposed in literature do not meet the stringent hardware constraints of current Tofino-based programmable switches, and therefore they cannot be entirely implemented in the switch pipeline. We are aware that, in general terms, BACON Sketch is a rather simple solution for per-flow cardinality estimation and more refined strategies have already been proposed. However, as pointed out earlier, BACON is the only solution that can be implemented in existing commodity programmable switches.

V. Potential integration of INDDoS in a full DDoS defense mechanism

In this section we explain how a DDoS defense system could benefit from the early detection of potential DDoS traffic in the data plane, as performed by INDDoS, as well as how the latter could benefit from the supervision of the former.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

A. Setting the DDoS detection threshold

The performance of INDDoS depends on selecting an appropriate threshold fraction θ, which determines the minimum number of sources contacting the same destination that trigger an alarm, discriminating between legitimate and abnormal traffic. In principle, to strike a good balance between false positives and false negatives, θ should be chosen so that θn *(i.)* is larger than the largest flow cardinality of any host in normal conditions (i.e., not under attack), and *(ii.)* is not so large that it fails to detect some attacks. Additionally, the threshold should be dynamically adapted to traffic changes.

Even though the definition of a detailed strategy for dynamic threshold setting is beyond the scope of this paper, we provide some indications on how it could be designed. In the simplest case, it could be set by mirroring and analyzing a large trace of legitimate traffic sampled in each switch. Alternatively, considering a more complex DDoS detection function at the controller, it could be set via a slow negative feedback control loop: start with a relatively high threshold, and slowly decrease it over time; at some point, the switch will start flagging some destinations as potentially under attack, but the controller could determine whether those are false positives, and if so raise the threshold back up. Finally, when θ changes, the BACON sketch size should be tweaked according to our theoretical analysis, in order to maximize the detection performance of INDDoS. In different deployment scenarios that attacks can traverse multiple disjoint paths, a controller could implement a network-wide strategy tweaking θ on the switches accordingly.

B. DDoS attack mitigation inside programmable switches

INDDoS enables the early detection of DDoS attack victims in the data plane pipeline that can be exploited to immediately mitigate them. Once INDDoS has identified a possible DDoS victim and sent a digest to the controller, two simple DDoS victim-based mitigation operations could be triggered in the data plane, i.e., *Drop* and *Rate limit*:

- **Drop**: the controller automatically configures, in all programmable switches in the network, the relevant egress Access Control List (ACL) with identified attack traffic features (e.g. source, destination IPs, ports) in order to drop the packets belonging to likely malicious flows.
- **Rate limit**: a meter is used to aggregate traffic towards the identified victim. If the rate is larger than a pre-defined threshold, the traffic rate is limited.

Although we already implemented both operations in P4, we are keenly aware that neither can actually implement proper DDoS mitigation alone. For that, external aid from the controller or an additional, more precise and computationally expensive detection mechanism (operating on the limited subset of suspicious flows identified by INDDoS) is needed, as outlined in the next subsection.

C. DDoS attack mitigation outside programmable switches

More refined and complex strategies, making use of external servers, have been proposed in literature for DDoS attack mitigation. Here we describe a few and explain how they can benefit from INDDoS.

Bohatei [22] is a strategy that spins up virtual machines (VMs) in servers for the identification, analysis and mitigation of suspicious traffic as needed: once suspicious traffic is detected, a resource manager determines the type, number, and location of VMs to be instantiated, and such traffic is steered to them for further analysis (and possibly mitigation) with minimal impact on users' perceived latency. Another approach, named Poseidon [23], performs DDoS mitigation by combining the capabilities of programmable switches and of sets of external servers. The additional mitigation functionalities provided by the servers, with respect to those provided by programmable switches, significantly improves the performance with respect to DDoS defense: Poseidon can mitigate sixteen different types of attacks exploiting different protocols (i.e., ICMP, TCP, UDP, HTTP).

However, both Bohatei and Poseidon consider a "DDoS-defense-as-a-service scenario", that is, they assume that the DDoS victim is known a-priori. INDDoS is complementary to these approaches since it can effectively detect and identify likely DDoS victims, and hence route only a manageable subset of network traffic towards these detection services.

VI. Experimental evaluation

We implemented INDDoS in a commodity Edgecore Wedge-100BF-32X switch equipped with Barefoot Tofino 3.3 Tbps ASIC [9], which supports up to 32 100 Gbps ports. Due to the high cost of 100 Gbps interfaces, we connected the switch to two servers (Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz, 16 GB RAM) using 10 Gbps Ethernet interfaces, as shown in Fig. 6. We also implemented a P4-based *simple forwarding* strategy for comparison purposes. Packets are sent across the switch via Tcpreplay [24].

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

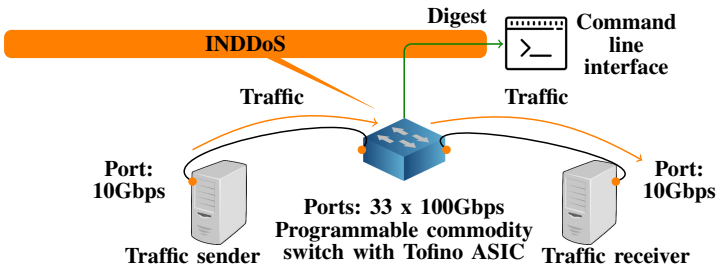


Fig. 6. The physical testbed for the experiments on DDoS victim identification

TABLE II
PROPERTIES OF DDOS FLOW TRACES [1]

DDoS trace name	Packets per second	Attack source IPs
Booter 6	∼ 90000	7379
Booter 7	∼ 41000	6075
Booter 1	∼ 96000	4486
Booter 4	∼ 80000	2970

A. Evaluation metrics and settings

1) Testing flow traces: In our first experiment (*Exp 1*) we used, as done in previous works on DDoS detection [5][25], a 50s passive CAIDA flow trace [15] collected from a 10Gbps backbone link, which we divided into 10 time intervals. Each 5s time interval contains around 2.3 million packets and 60 thousand distinct source IPs. Note that, although the trace did not contain actual DDoS traffic, we were still able (by using a low, sensitive threshold) to successfully detect outliers, despite the number of incoming connections for such cases being far smaller than in actual volumetric DDoS scenarios. In other words, our experiment is a pressure test: if INDDoS can detect such outliers accurately, by properly setting a threshold between legitimate and DDoS traffic, it should have little trouble in properly distinguishing actual attack traffic where such difference is more marked.

In a second experiment (*Exp 2*) we then considered real DDoS attacks, namely from Booter [1]. Booter is a class of on-demand services that provide illegal support to launch DDoS attacks targeting websites and networks. We considered four 50s Booter DNS-amplification DDoS attack traces (i.e. booter 1, 4, 6, 7), which are the ones with the highest number of attack source IPs among all the available traces in [1]. This allows us to investigate the other extreme with respect to just considering the CAIDA trace without attacks (Exp 1). Table II reports the salient properties of the chosen traces. We split each of them into 10 time intervals as for the CAIDA trace, and appended each 5s attack trace at the end of its respective legitimate trace. The attack targets are the destinations specified in the Booter traces (one per trace), and the attacked destination IPs are not present in the CAIDA normal traffic. In this way we generated four traces containing a single attack, as well as a fifth *Mixed* trace containing all four attacks simultaneously along with the legitimate CAIDA traffic. Note that in typical DNS-amplification DDoS attacks the number of usable amplifying reflectors is smaller than that of bots and usable reflectors: this means that we are considering the lowest number of source IPs that might be expected from a volumetric (non-semantic) attack, and thus we are not posing ourselves in the most favorable condition (i.e., with the strongest possible outliers).

2) Metrics: We chose *recall* R and *precision* Pr as the key metrics to evaluate INDDoS, defined as follows:

$$R = \frac{Count_{DDoSvictim}^{detected/true}}{Count_{DDoSvictim}^{detected/true} + Count_{DDoSvictim}^{undetected/true}}$$

$$Pr = \frac{Count_{DDoSvictim}^{detected/true}}{Count_{DDoSvictim}^{detected/true} + Count_{DDoSvictim}^{detected/false}}$$

The count of DDoS victims is obtained via the command line interface provided by our Tofino-based switch.

In our evaluations, we considered the well known *F1 score* ($F1$) as a compact metric incorporating both precision and recall, and thus measuring the accuracy of our strategy. It is defined as:

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

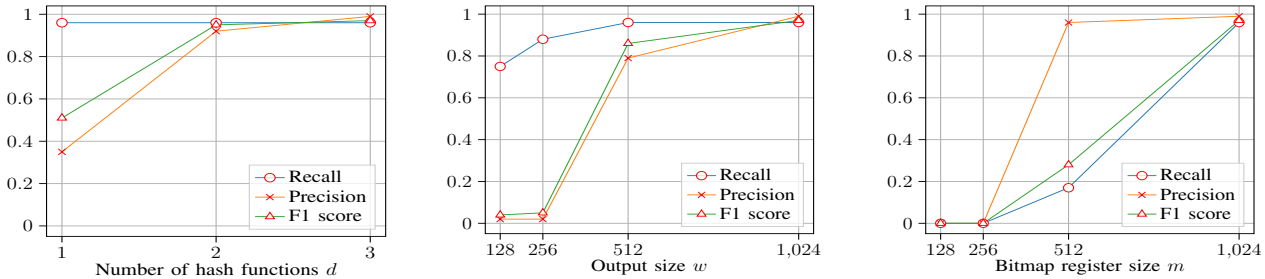
All results are mean values computed across 10 time intervals.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

TABLE III
COMPARISON OF INDDOS PERFORMANCE AS A FUNCTION OF BACON SKETCH PARAMETERS AND WITH SPREAD SKETCH [20]

BACON Sketch size ($d \times w \times m$)	**Recall**	**Precision**	**F1 score**
$3 \times 1024 \times 1024$	0.96	0.99	0.97
$1 \times 2048 \times 1024$	0.98	0.54	0.70
$1 \times 1024 \times 2048$	0.94	0.38	0.54
$5 \times 1024 \times 512$	0.12	1.0	0.22
$5 \times 512 \times 1024$	0.96	0.89	0.92
Spread Sketch size ($d \times w \times m$)	**Recall**	**Precision**	**F1 score**
$3 \times 1024 \times 1024$	0.92	0.94	0.93



(a) Sensitivity analysis to d (b) Sensitivity analysis to w (c) Sensitivity analysis to m

Fig. 7. Sensitivity analysis of INDDoS to the parameters of BACON Sketch

3) Tuning parameters: The default BACON Sketch size $d \times w \times m$ is $3 \times 1024 \times 1024$, and the DDoS threshold fraction θ is set to 0.5% as per [5] in the first experiment, and to 1% in the second, to better separate legitimate and DDoS traffic.

B. Exp 1: evaluation of DDoS victim identification accuracy

According to Theorems 1 and 2, the Bitmap register size m impacts both upper and lower bounds on the quality of BACON Sketch's estimation: increasing m increases the distance between the upper bound of the estimation and the true flow cardinality, but also reduces that between the lower bound and the true value. This implies that m should be neither too large nor too small. In our flow trace, the largest flow cardinality in each time interval is within $[2^9, 2^{10}]$, so we set m to 1024. Conversely, larger output sizes w generate lower false negative bounds and do not affect the false positive bound, but in order to apply the false positive bound proved in Theorem 4, w should satisfy $w \geq \frac{4}{\theta}$. Therefore, given $\theta = 0.5\%$ in our case, w should be larger than $\frac{4}{0.5\%} = 800$. However, due to hardware limitations, we cannot use arbitrarily large values for w; in fact, the only feasible parameter combinations that satisfy all conditions on m and w, while not exceeding the available pipeline stages, are: $3 \times 1024 \times 1024$ and $1 \times 2048 \times 1024$. Considering that $1 \times 2048 \times 1024$ only uses 1 hash function, the estimated flow cardinality has a high chance (i.e., $\frac{1}{2}$) to be out of both upper bound and lower bound. Note that the number of hash functions d = 2 was not considered since 2 hash functions support the same largest register size (i.e., $w \times m$ = 1024×1024) as 3 functions within the available pipeline stages. Likewise, 4 hash functions support the same largest register size (i.e., $w \times m$ = 512×1024) as 5 functions, but neither can satisfy Theorem 4. Thus, according to the theoretical analysis, the best choice of the sketch size should be $3 \times 1024 \times 1024$ (hence our default choice).

Table III shows the performance of INDDoS for different BACON Sketch sizes. In all cases listed in the table, all available pipeline stages are required to implement the strategy, while the occupied memory is much lower than available switch's memory. Results show good performance on both recall and precision for $3 \times 1024 \times 1024$, which translates to a high F1 score. With only one hash function in BACON Sketch, INDDoS shows high recall, i.e., it is able to identify most DDoS victims, but low precision, meaning that a lot of false identifications take place. When instead $m = 2048$, the recall of $1 \times 1024 \times 2048$ is as high as for $1 \times 2048 \times 1024$, but its precision decreases by a third. In contrast, when $m = 512$ and the sketch size is $5 \times 1024 \times 512$, the recall significantly decreases leading to a much lower F1 score than for $5 \times 512 \times 1024$. These two results indicate that a larger m causes higher recall but lower precision, and vice versa. The recall for sketch size $3 \times 1024 \times 1024$ is comparable to the one for size $5 \times 512 \times 1024$, but the precision is 10 percentage points higher. Note that in higher-speed networks, where the largest flow cardinality during a time interval may exceed 1024, INDDoS can still work properly (even in currently available switches) by shrinking the time interval.

1) Comparison with the state of the art: We considered SpreadSketch [20] as an alternative sketch for INDDoS rather than BACON Sketch and compared the DDoS detection performance in terms of simulations. SpreadSketch is three-dimensional as BACON Sketch, and we set the same size of $3 \times 1024 \times 1024$ for the two approaches. The results are shown in Table

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

TABLE IV
COMMUNICATION OVERHEAD COMPARISON

Strategy	**Communication overhead (per T_{int})**
INDDoS (BACON Sketch)	$(4 \cdot N_{victim})$B
SpreadSketch ($3 \times 1024 \times 1024$)	0.375MB
NetFlow (sampling rate 1/2000)	1.15MB

III. When BACON Sketch is adopted, both recall and precision are slightly higher, and therefore the overall detection performance (i.e., F1 score) is also better. This is because the query operation in SpreadSketch uses Multiresoultion Bitmap for flow cardinality estimation, while BACON Sketch uses Direct Bitmap. When there are not many collisions in the sketch, the Direct Bitmap adopted by BACON Sketch leads to slightly higher accuracy than Multiresoultion Bitmap. Additionally, the great advantage of BACON Sketch is that it can be fully executed in the data plane (for both update and query operations), while the query operation of SpreadSketch can only be executed by the controller. This leads to great advantages in terms of communication overhead and detection speed, as it will be shown in Section VI-D.

C. Exp. 1: sensitivity analysis of DDoS victim identification

In order to show how the performance of INDDoS is sensitive to different tuning parameters (number of hash functions d, output size w and Bitmap register size m) in BACON Sketch, we conducted some experiments by varying tuning parameters individually, while respecting the resource constraints of our switch. The results are reported in Fig. 7.

1) Sensitivity to number of hash functions d: Fig. 7(a) shows how INDDoS behaves with d ranging from 1 to 3. Recall is high in all three cases, but precision increases as d increases. This is because more hash functions lead to smaller number of collisions in the Count-min Sketch side of BACON Sketch, and hence to a lower overestimation of the number of source IPs contacting a specific destination.

2) Sensitivity to output size of hash functions w: Fig. 7(b) shows how INDDoS performs by varying w from 128 to 1024. Clearly, precision is very sensitive to w, and it significantly decreases as w decreases. Conversely, even for small values of w (e.g., $w = 128$), recall kkkkins above 0.75.

3) Sensitivity to Bitmap register size m: Fig. 7(c) shows how INDDoS performance changes along with the value of m. Since the threshold in each time interval is θn, which is around $0.5\% \cdot 60000 = 300$ in our evaluation, when m is below 300, i.e., $m \in \{128, 256\}$, no victim can be detected and thus the F1 score is 0. Precision of $m = 512$ and of $m = 1024$ is comparable, but recall is much lower for $m = 512$. As proven theoretically, a smaller m causes a smaller lower bound on the flow cardinality estimation $\hat{E}_{dst}$, so it is more difficult for $\hat{E}_{dst}$ to exceed the threshold.

D. Exp. 1: communication overhead and detection speed comparison with the state of the art

In this subsection, we compare INDDoS with two state-of-the-art approaches, i.e., SpreadSketch [20] and Netflow [7], in terms of communication overhead and detection speed. SpreadSketch is queried by the controller, while NetFlow is used to collect packet statistics at the controller side; in both cases, the DDoS detection logic is executed by the controller.

Table IV reports a comparison in terms of communication overhead between the considered strategies with focus on a single time interval T_{int}. The sketch size of BACON Sketch and SpreadSketch is set to $3 \times 1024 \times 1024$ (see Section VI-B). However, the two approaches lead to very different overheads. INDDoS only reports detected victim IP addresses to the controller: considering that each IP address is 4 bytes long, the communication overhead is $(4 \cdot N_{victim})$B, where N_{victim} is the number of detected DDoS victims by INDDoS in T_{int}. Instead, SpreadSketch needs to send the whole sketch to the controller, at the end of T_{int}, for further processing (i.e., query on the sketch) and victim identification. As each SpreadSketch cell occupies 1 bit, the overall size of the sketch is $\frac{3 \cdot 1024 \cdot 1024}{8}$B = 0.375MB. Concerning NetFlow, the considered sampling rate is 1/2000, being the recommended value for 10Gbps link speed [26] and the maximum rate not affecting DDoS detection accuracy [27]. The average total traffic in each 5s time interval, for the considered flow trace, is 2.3GB, meaning that the average sampled communication overhead is around $\frac{2300}{2000}$MB $=$ 1.15MB. With respect to the two existing approaches, the communication overhead generated by INDDoS is almost negligible, given that N_{victim} is expected to be on average low in any T_{int}. If no victim is identified (i.e., $N_{victim} = 0$) no communication is needed between the programmable switch and the controller.

Concerning detection speed, INDDoS reports to the controller immediately once a victim is detected and identified (see Fig. 4). Conversely, the other two methods require the execution of the DDoS detection logic by the controller at the end of every T_{int}: clearly, the longer T_{int} is, the faster (on average) our approach is with respect to the state of the art.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

TABLE V
INDDOS PERFORMANCE FOR BOOTER DDOS ATTACKS

DDoS attack flow trace	Recall	Precision	F1 score
Booter 6	1.0 (1/1)	1.0 (1/1)	1.0 (1/1)
Booter 7	1.0 (1/1)	1.0 (1/1)	1.0 (1/1)
Booter 1	1.0 (1/1)	1.0 (1/1)	1.0 (1/1)
Booter 4	1.0 (1/1)	1.0 (1/1)	1.0 (1/1)
Mixed	1.0 (4/4)	1.0 (4/4)	1.0 (4/4)

TABLE VI
NETWORK PERFORMANCE OF INDDOS IN THE COMMODITY SWITCH

Type	iPerf size	Throughput	Jitter	Packet loss	Average additional processing time w.r.t. simple forwarding
TCP	64 KB	9.02 Gbps	/	/	106 ns
TCP	128 KB	9.41 Gbps	/	/	101 ns
UDP	500 B	1.03 Gbps	0.003 ms	0%	102 ns
UDP	1470 B	2.87 Gbps	0.004 ms	0%	102 ns
UDP	3000 B	5.45 Gbps	0.004 ms	0%	107 ns
UDP	6000 B	9.62 Gbps	0.004 ms	0%	104 ns
UDP	9000 B	9.67 Gbps	0.006 ms	0%	101 ns

E. Exp. 2: evaluation of DDoS victim identification accuracy under Booter DDoS attacks

Table V shows the performance of INDDoS on victim identification under actual DDoS attacks with different number of attack source IPs. The threshold is approximately $60000 \cdot 1\% = 600$, which is larger than the largest flow cardinality for legitimate traffic in each 5s interval. Using these parameters, INDDoS can always correctly identify the victim even though the number of attack source IPs in all cases is greater than the Bitmap size (1024), since our threshold is lower than that and a saturated count (i.e., all 1s in the Bitmap register) still results in flagging a destination as potential DDoS victim. Even though there are four distinct DDoS attacks in the Mixed case, INDDoS correctly identifies all four different victims. This suggests that, when a suitable threshold is used (insights on how to set it are given in Section V), INDDoS can identify victims almost perfectly.

Furthermore, we observe that in our CAIDA flow trace the maximum flow cardinality in 5s time intervals is around 500, whereas in 1s intervals it is about 260. Therefore, shrinking the time interval not only allows INDDoS to detect attacks on faster networks, but it also increases the difference between legitimate and DDoS traffic, simplifying the selection of a threshold and increasing detection accuracy.

F. Evaluation of impact on network performance

We used iPerf3 [28] to indirectly measure the performance of INDDoS in the switch. The results are reported in Table VI. We first generated 10 Gbps of TCP traffic with iPerf buffer size 64 KB and 128 KB from one server to the other across the switch. In both cases, the throughput reached more than 9 Gbps, i.e., the TCP traffic could be processed at line-rate (with 10 Gbps interfaces). We then generated 10 Gbps of UDP traffic with different iPerf sizes ranging from 500B to 9000B. Using datagrams smaller than 1470B (i.e., the typical value for home access) our server could not reach 10Gbps, being the server's CPU processing capacity a bottleneck in this settings. Therefore, we increased the *maximum transmission unit* (MTU) to allow UDP datagram sizes larger than 1470B. The results show that the throughput and jitter increased as packet size increased, but without packet loss. When the datagram size reached 6000B, the throughput was 9.62Gbps, indicating that INDDoS could also process UDP datagrams at line-rate. Although our testbed limitations did not allow us to test the switch behaviour at high packet processing rates, previous works [29] strongly suggest that performance would not be affected by INDDoS, as for any strategy that can be fully compiled and executed by the Tofino hardware.

Additionally, we embedded two registers in our P4 program to monitor packet processing time. One was placed at the ingress pipeline, and stored the timestamp t_{in} of when a packet entered the switch. The timestamp t_{out} was recorded in another register at the egress pipeline when the packet had been processed. Per-packet processing time could thus be calculated as $t_{out} - t_{in}$. This averaged around 200 ns for simple forwarding, with INDDoS adding an additional 100 ns or so on top of that, which we deem an acceptable time overhead.

G. Evaluation of resource usage

Table VII shows the switch resources required by our INDDoS and simple forwarding implementation. Considering that each stage can only apply a single atomic action, several stages must be used to carry out INDDoS calculations including

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

TABLE VII
SWITCH RESOURCE USAGE OF INDDOS

Strategy	No. stages	SRAM	No. ALUs	PHV size
Simple forwarding	16.67%	2.5%	4.2%	7.30%
INDDoS + Simple forwarding	100%	8.33%	56.25%	9.90%

simple forwarding, meaning that we require all available stages in the switch to achieve the best detection performance, whereas simple forwarding only requires a small fraction of them (16.67%). Moreover, our strategy uses only 8.33% of the total available SRAM. This is because each register cell in BACON Sketch only occupies 1 bit, and the size of Count-min Sketch is small (i.e., only 3×1024) though each of its register cells occupies 32 bits. INDDoS needs 56.25% of total ALUs for processing the packets. This is high since INDDoS not only needs to apply the match-action table for simple packet forwarding (4.2% of all ALUs), but also maintains two sketches (BACON and auxiliary), which require more arithmetic operations. The packet header vector (PHV) size indicates the amounts of packet header information passed across the pipeline stages. In our case, only 9.90% of PHV is required for storing this information, compared to 7.30% for simple forwarding, meaning that INDDoS does not embed much additional temporary metadata in the packet.

It is important to clarify that, even though INDDoS requires all available pipeline stages, it does not consume all the resources in each stage (as shown in the columns SRAM, No. ALUs and PHV size). This means that it would be possible to consolidate other tasks (e.g. DDoS mitigation as introduced in Section V-B) in the same pipeline if properly developed. Moreover, the relative cost of our solution is expected to be lower if deployed in next-generation programmable hardware (e.g. Tofino 2), which will provide more resources and stages.

VII. RELATED WORKS

In this section we describe recent related works and solutions on *(i.)* DDoS detection in Software-Defined Networks, *(ii.)* programmable data plane capabilities with ASICs and *(iii.)* in-network monitoring using programmable switches.

A. DDoS detection in the context of SDN

Many techniques have already been proposed to detect various kinds of DDoS attacks in SDN networks. A DDoS attack can be identified according to many different metrics, such as looking for a significant decrease of the normalized entropy in distinct destination IP addresses observed in the network [30][31][32], or a large number of distinct flows (sequences of packets with the same source IPs) contacting a specific destination host (i.e., per-destination flow cardinality) [3][4][5]. Note that entropy-based DDoS detection can only detect DDoS attacks, but flow cardinality-based DDoS detection is also able to identify the DDoS victims, which allows operators to mitigate the impact on targeted nodes as soon as an attack is detected. However, the state-of-the-art approaches [3][4][5] in SDN still need the controller to periodically retrieve the information from the switches for further processing. With our flow-cardinality-based INDDoS approach we make a step further: we exploit data-plane programmable switches to just forward the information on DDoS victims to the controller, offloading the DDoS detection task to the switch.

B. Data plane programmable switches exploiting ASICs

Most of the existing SDN switches come with very limited (or no) programmability with respect to the data plane functions that can be executed. To enable new kinds of functionalities (e.g. support of additional protocols) it is necessary to upgrade the hardware, which requires significant capital expenditure. Recently, programmable ASICs have been introduced: they ensure standard data plane features (i.e., high-speed switching and forwarding) while offering the possibility of customizing functionalities, if properly programmed through domain-specific programming languages like P4 [10]. For instance, programmable switches equipped with Tofino ASIC [9] can always forward packets at line-rate once the P4 program (including innovative features) is compiled and installed in the switches. Other programmable chips, like Network Interface Cards (NICs), Field Programmable Gate Arrays (FPGAs) and Network Processing Units (NPUs), cannot currently ensure high throughput and low latency on par with ASICs. Additionally, in the context of network security, compared to highly-optimized software solutions, such as inline Intrusion Detection Systems (IDSs) [33], the throughput ensured by ASICs is orders of magnitude higher and introduces much lower latency ($\sim 50\mu s - 1ms$) [23]. This makes programmable ASICs well suited for the implementation of some network monitoring/security tasks, such as the DDoS detection strategy proposed in this paper.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

C. In-network monitoring tasks using programmable switches

Network monitoring has been widely studied, including in the context dealing with the capability of programmable switches. Recently, researchers have started embedding network monitoring tasks directly into programmable switches, such as heavy hitter detection [34], network traffic entropy estimation [21] and entropy-based DDoS detection [35]. Most of these solutions are based on sketches, probabilistic data structures to track summarized information pertaining large numbers of packets using fixed size memory. It has been proven that sketch-based monitoring solutions have a better accuracy/memory trade-off than sampling-based solution, at least over short time scales [36]. A common feature of these approaches is that the monitoring outcomes gathered from sketches are reported to the controller only when an anomalous event is detected, therefore overcoming the limitations of large communication overhead and latency caused by the interaction between data plane and control plane. Unfortunately, another common theme among these works is that, unlike ours, their P4 code was only tested in the (largely resource-unconstrained) P4 Behavioral model [12] simulator. Exceptions to this include Tang *et al.* [20], who proposed SpreadSketch, a new sketch for DDoS detection implementable in Tofino-based switches. However, their solution relies on the controller querying the sketch for identifying the attacks, while in our proposal queries occur inside the switch, thus reducing switch-controller interaction needs. Dimolianis *et al.* [37] presented another in-network DDoS detection scheme working in actual Netronome SmartNICs [38]. Their approach measures three different features: total number of incoming traffic flows, subnet significance and packet symmetry. However, it is only able to identify the subnet under attack, which may limit the accuracy of deployed mitigation measures.

To the best of our knowledge, INDDoS is the first attempt to perform DDoS detection with host victim identification entirely in commodity switches equipped with programmable ASICs, while dealing with and overcoming all the constraints set by the hardware.

VIII. Conclusions

In this paper, we proposed a novel in-network DDoS victim identification strategy, INDDoS, based on a new probabilistic data structure we named BACON Sketch, which combines a Direct Bitmap and a Count-min Sketch to estimate the number of distinct flows contacting the same destination. INDDoS uses a threshold-based rule to identify victims directly in the programmable data plane of switches. We proved some parametric bounds on the quality of estimations produced by BACON and INDDoS, and implemented them using the P4 language and toolchain in a Edgecore commodity switch with Tofino ASIC. The analysis of the performance of our solution proves that it can precisely and accurately identify DDoS victims without adversely affecting the packet processing capabilities of the switch. Moreover, this approach only reports to the controller when a new victim is detected, greatly reducing the communication strain on the monitoring infrastructure. This work contributes to the ongoing DDoS attack detection and mitigation activities carried on in the GN4-3 project for upgrading the pan-European GÉANT network.

Acknowledgement

The research leading to these results has received funding from the GN4-3 project, within the European H2020 R&I program, Grant Agreement No. 856726. We also want to thank Barefoot Networks for their valuable support.

References

[1] J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Zambenedetti Granville, and A. Pras, "Booters - An analysis of DDoS-as-a-service attacks," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015.
[2] S. Ramanathan, J. Mirkovic, M. Yu, and Y. Zhang, "SENSS against volumetric DDoS attacks," in *Annual Computer Security Applications Conference*, 2018.
[3] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with OpenSketch," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013.
[4] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2016.
[5] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "SketchVisor: Robust network measurement for software packet processing," in *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2017.
[6] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network monitoring in OpenFlow software-defined networks," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2014.
[7] "Cisco IOS NetFlow," https://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/ios-netflow.
[8] D. Ding, M. Savi, G. Antichi, and D. Siracusa, "An incrementally-deployable P4-enabled architecture for network-wide heavy-hitter detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 75–88, 2020.
[9] "Tofino," https://www.barefootnetworks.com/products/brief-tofino/.
[10] P. Bosshart, D. Daly, G. Gibb *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.
[11] K. Qian, S. Ma, M. Miao, J. Lu, T. Zhang, P. Wang, C. Sun, and F. Ren, "FlexGate: High-performance heterogeneous gateway in data centers," in *Asia-Pacific Workshop on Networking (APNet)*, 2019.
[12] "P4 behavioral model," https://github.com/p4lang/behavioral-model.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.

This is the author's version of an article that has been published in IEEE Transactions on Network and Service Management. Changes were made to this version by the publisher prior to publication. The final version of record is available at 10.1109/TNSM.2021.3073597.

[13] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *ACM SIGCOMM conference on Internet measurement*, 2003.
[14] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the Count-min Sketch and its applications," *Elsevier Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.
[15] "CAIDA UCSD anonymized internet traces dataset - [passive-2018]," http://www.caida.org/data/passive/passive_dataset.xml.
[16] "GÉANT network," http://https://www.geant.org/.
[17] S. Yu, W. Zhou, W. Jia, S. Guo, Y. Xiang, and F. Tang, "Discriminating DDoS attacks from flash crowds using flow correlation coefficient," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 1073–1080, 2011.
[18] "P4 code of INDDoS," https://github.com/DINGDAMU/INDDoS.
[19] Q. Xiao, S. Chen, Y. Zhou, M. Chen, J. Luo, T. Li, and Y. Ling, "Cardinality estimation for elephant flows: a compact solution based on virtual register sharing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3738–3752, 2017.
[20] L. Tang, Q. Huang, and P. P. Lee, "SpreadSketch: Toward invertible and network-wide detection of superspreaders," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2020.
[21] D. Ding, M. Savi, and D. Siracusa, "Estimating logarithmic and exponential functions to track network traffic entropy in P4," in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2020.
[22] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDoS defense," in *USENIX Security Symposium*, 2015.
[23] M. Zhang, G. Li, S. Wang, C. Liu, A. Chen, H. Hu, G. Gu, Q. Li, M. Xu, and J. Wu, "Poseidon: Mitigating volumetric DDoS attacks with programmable switches," in *Network and Distributed System Security Symposium (NDSS)*, 2020.
[24] "Tcpreplay," https://tcpreplay.appneta.com/.
[25] N. N. Tuan, P. H. Hung, N. D. Nghia, N. V. Tho, T. V. Phan, and N. H. Thanh, "A DDoS Attack Mitigation Scheme in ISP Networks Using Machine Learning Based on SDN," *Electronics*, vol. 9, no. 3, 2020.
[26] "NetFlow Sampling Rate," https://confluence.netvizura.com/pages/viewpage.action?pageId=27885662.
[27] G. Roudière and P. Owezarski, "Evaluating the Impact of Traffic Sampling on AATAC's DDoS Detection," in *ACM Workshop on Traffic Measurements for Cybersecurity*, 2018.
[28] "iPerf3," https://iperf.fr/iperf-doc.php.
[29] Y. Zhou, Z. Xi, D. Zhang, Y. Wang, J. Wang, M. Xu, and J. Wu, "HyperTester: High-Performance Network Testing Driven by Programmable Switches," 2019.
[30] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments," *Elsevier Computer Networks*, vol. 62, pp. 122–136, 2014.
[31] K. Kalkan, L. Altay, G. Gür, and F. Alagöz, "JESS: Joint entropy-based DDoS defense scheme in SDN," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2358–2372, 2018.
[32] R. Wang, Z. Jia, and L. Ju, "An entropy-based distributed DDoS detection mechanism in software-defined networking," in *IEEE Trustcom/BigDataSE/ISPA*, 2015.
[33] T. Krueger, C. Gehl, K. Rieck, and P. Laskov, "An architecture for inline anomaly detection," in *European Conference on Computer Network Defense*, 2008.
[34] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *ACM Symposium on SDN Research (SOSR)*, 2017.
[35] A. C. Lapolli, J. A. Marques, and L. P. Gaspary, "Offloading real-time DDoS attack detection to programmable data planes," in *IEEE/IFIP Symposium on Integrated Network and Service Management (IM)*, 2019.
[36] 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 *ACM Special Interest Group on Data Communication (SIGCOMM)*, 2018.
[37] M. Dimolianis, A. Pavlidis, and V. Maglaris, "A multi-feature DDoS detection schema on P4 network hardware," in *Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, 2020.
[38] "Netronome SmartNICs," https://www.netronome.com/products/agilio-cx/.

Copyright (c) 2021 IEEE. This work is made available online in accordance with the publisher's policies.
Please refer to any applicable terms of use of the publisher.