In-Kernel Traffic Sketching for Volumetric DDoS Detection

Mingyuan Zang*, Federico De Iaco†, Jie Wu‡, Marco Savi†
*China Telecom Cloud Computing Research Institute, China
‡Department of Computer and Information Sciences, Temple University, USA
†Department of Informatics, Systems and Communication (DISCo), University of Milano-Bicocca, Italy

***Abstract*—Emerging network technologies like cloud computing provide flexible services but also introduce vulnerabilities to host servers, such as exposure to Distributed Denial of Service (DDoS) attacks. Traditional host-based detection tools operate in the user space, which can delay detection because network traffic must pass through the kernel space first. Moving detection to the kernel space speeds up the process but limits traffic processing capabilities. We propose using a memory-efficient sketch-based data structure for kernel-space DDoS detection, allowing attacks to be identified immediately upon arrival at the host. We also introduce double-sketch and adaptive-threshold mechanisms to circumvent some inefficiencies of eBPF and to optimize this design for dynamic network traffic. Experiments in a controlled environment show that our approach achieves over 90% detection performance and accelerates it to a sub-microsecond level compared to the millisecond level in classical user-space methods.**

***Index Terms*—eBPF, DDoS Detection, Kernel Space, Sketching.**

I. Introduction

Distributed Denial-of-Service (DDoS) attacks have become a major concern for network operators and their customers as these attacks, launched by numerous compromised devices, can overwhelm a target with excessive traffic, exhausting its network bandwidth and computational resources and leading to service interruptions. In modern networks, the number of DDoS attacks is constantly increasing, rendering their efficient and effective detection and identification pivotal [1]. However, the recent massive adoption of virtualization technologies both within and at the edge of the network makes such an environment more diverse and dynamic, making the aforementioned detection activity more challenging [2].

Regularly monitoring some specific network traffic metrics has become the fundamental approach to achieve this goal: for instance, a sharp rise in the number of source IPs contacting a particular destination IP can be an indicator of an in-progress DDoS attack [3]. To mitigate the impact of such volumetric attacks on other parts of the network, attack detection systems deployed on network devices (or middleware) at the network edge have been proposed as first-line of defense. While edge-based systems can provide network-wide detection, the latter is mainly concerned with filtering malicious traffic before it reaches internal systems. When it comes to threats within an edge-cloud computing environment, such as lateral movement attacks [4], edge-based solutions show their limitations.

To also mitigate DDoS caused by internal threats, host-based detection solutions are very promising. Host machines such as servers in data centers have a *user space* and a *kernel space*. While user space is the area where user applications and processes run, kernel space is a lower-level area where the core of the operating system executes: classical host-based solutions are deployed as applications in the user space, but such methods experience latency [5] as the network traffic has to first go through the kernel space and then be forwarded to the detection application in the user space for inspection.

Recent advancements in the operational capability of Linux kernels by means of the *extended Berkeley Packet Filtering* (eBPF) technology have provided new possibilities. By exploiting kernel-space solutions such as the aforementioned eBPF and the related *eXpress Data Path* (XDP) [6], high-performance packet processing and monitoring tasks can be moved from user to kernel space. This type of offloading can enable an early inspection of network traffic by avoiding the necessity to forward it from kernel to user space. However, there are still well-known constraints in the kernel space that limit the detection performance [7].

The main focus of this paper is to guarantee an efficient and effective *in-kernel* network monitoring on hosts for the detection of *volumetric* DDoS attacks. The primary goal is to provide a high-performance *traffic sketching* solution in eBPF/XDP while ensuring that both *Query* and *Update* operations performed by the algorithm are executed in kernel space by the eBPF program, without any involvement of user-space logic. The sketching algorithm is inspired by an existing DDoS detection algorithm [3], which has been proven efficient and effective on network P4 switches but not on hosts. To address differences between switches and host machines, we adapt the algorithm for kernel-space design. With respect to an implementation where the algorithm completely or partially relies on a user-space application, a full adoption of eBPF/XDP greatly accelerates packet processing performance – and thus reaction to DDoS attacks – as it avoids unnecessary user-kernel space interactions, which are computationally expensive [5]. In addition, relying on a sketching algorithm and related data structure has the benefit, with respect to existing alternatives (e.g. [8] [9]) of keeping the memory consumption bounded.

To the best of our knowledge, this is the first attempt to provide a solution for host-based DDoS detection that combines the advantages of eBPF and traffic sketching, while also exploring and overcoming the limitations of such a technology [7] to ensure enhanced performance. Specifically, our proposal relies on a *double-sketch mechanism* to ensure seamless monitoring also when the sketch needs to be reset and parameters updated (i.e., between consecutive time windows), which has been experienced to take a non-negligible amount

of time. To enhance detection performance, we also propose the adoption of a *self-adaptive dynamic threshold* that is able to re-adjust to ever-changing network traffic patterns.

Our experimental results show that our solution is effective in detecting DDoS attacks while keeping the amount of consumed memory low (i.e., in the order of few megabytes). In addition, our implementation choices make our solution several orders of magnitude faster with respect to adopting a similar solution fully relying on a user-space program.

Our contributions in this work can thus be summarized as:

- We propose an eBPF-based sketching data structure for fast DDoS attack detection in end hosts;
- We propose a novel double-sketch design to mitigate the limitations of eBPF and optimize the performance of in-kernel sketch-based detection;
- We integrate self-adaptive thresholds into the eBPF sketch design to ensure accurate detection at runtime even in the face of dynamic traffic intensities.

II. Background

A. Extended Berkeley Packet Filtering and eXpress Data Path

eBPF is a powerful Linux technology enabling the safe and efficient execution of sandboxed programs directly in the kernel without recompilation. Key components of eBPF include: *(i) eBPF programs*, which are user-defined code lines (usually written in C language), compiled to bytecode, and loaded into the kernel; *(ii) maps*, which are efficient data structures used for communication between user space and eBPF programs, storing data like counters or packet statistics; *(iii) hooks*: which are attachment points in the kernel, such as XDP, where eBPF programs can be triggered to process events like packet arrivals. The adoption of eBPF and XDP enables flexible data-plane logic with high efficiency, capable of processing 20-25 million packets/s on a single CPU core [10].

B. Sketching algorithms: Direct Bitmap and Count-min Sketch

Our solution is based on BACON Sketch [3], which relies on a combination of two well-known sketching algorithms and related data structures: *Direct Bitmap* and *Count-min Sketch*.

Direct Bitmap [11] is used for estimating *flow cardinality* (i.e., number of distinct flows) in packet streams. It uses an m-sized bit array (*Bitmap register*) and one or more *hash functions*. Initially, all m cells of the register are set to 0. A packet's flow key is hashed to select a cell, which is set to 1 if it was previously 0. Packets with the same key hash to the same cell, while different flows hash to different cells. Flow cardinality is estimated by the number of 1s in the register.

The *per-flow packet count* can instead be estimated using a *Count-min Sketch* [12]. It supports two operations: *Update*, which tracks incoming packets in the node, and *Query*, which retrieves the estimated packet count for a specific flow. Count-min Sketch uses d pairwise-independent *hash functions* and a $d \times w$ matrix of counters. Accuracy improves as d or w increase, allowing more precise packet count estimations.

III. Related Work

A. DDoS detection and mitigation with eBPF/XDP

In literature some works adopting eBPF/XDP for the fast detection and mitigation of DDoS attack in host-based systems can be found. Ref. [14] is a seminal work on the topic, proposing a DDoS detection and mitigation strategy where part of the computation is executed in the host's kernel space with eBPF/XDP and part is offloaded to SmartNICs specialized hardware. Similarly to Ref. [14], Ref. [13] proposes a SYN Flood attack detection strategy relying on shared maps between kernel and user spaces. Maps are populated by a DDoS detection algorithm that, also in this case, runs in user space. In both works, all the detection (and mitigation) logic is executed in user space, while we aim at offloading them to the kernel space to enhance performance.

TABLE I
Comparison with related work

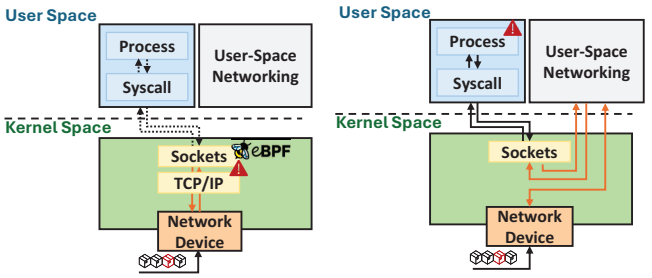
Ref.	Deployment Location	Packet Proc. Location	Attack	Detector Threshold	Resp. Time
[13]	NIC	Kernel/User	SYN Flood	Static	High
[8]	Host (k8s)	Kernel/User	DoS	Static	High
[3]	Switch	User	DDoS	Dynamic	High
Ours	Host/NIC	Kernel	DDoS	Dynamic	Low

In Ref. [15], a system using eBPF/XDP is developed for detecting and mitigating water torture DDoS attacks on authoritative DNS servers, utilizing a Bloom Filter, stored in eBPF maps, for filtering invalid requests populated by a user space program. Our approach is different as both the *Query* and *Update* of the BACON Sketch [3] occur in kernel space, avoiding interactions with the user space. Ref. [16] introduces a distributed proxy-based system for IoT networks employing eBPF/XDP to detect abnormal packet frequencies during DDoS attacks. It relies on a centralized database for uniform visibility across proxies, unlike our node-independent proposal. In the IoT domain, Ref. [17] integrates the MUD standard with eBPF to shape IoT traffic and prevent DDoS attacks. Ref. [8] uses eBPF/XDP to protect Kubernetes (k8s) clusters by monitoring connection counts from IP addresses, but faces scalability issues with high memory demands in the case of many malicious IPs. Similarly, Ref. [9] captures network packets in kernel space to detect volumetric attacks before reaching the user space. However, it also struggles to scale with multiple IPs. Our work addresses this by using a memory-efficient BACON Sketch to summarize data from many malicious sources without high memory requirements. Table III reports a more detailed comparison between our work and the most relevant existing works.

B. Sketching algorithms in eBPF/XDP

In the recent years, some works have focused on the implementation of sketches in eBPF for widely different applications. Ref. [18] proposes a resource-efficient network monitoring architecture based on eBPF/XDP. One of its peculiarities is the adoption of a HyperLogLog sketch stored in maps for estimating the number of unique flows (i.e., the cardinality). The sketch is updated by an eBPF program, while the cardinality is estimated by means of a *Query* operation by a user space program. In our work we instead implement both *Query* and *Update* operation of the BACON Sketch [3] in kernel space, thus making the query operation more efficient.

Ref. [7] focuses on identifying the best practices for sketches implementation in eBPF, with the goal of enhancing the performance of sketching in kernel space. The authors



(a) Our in-kernel design. (b) Classical design in user space.

Fig. 1. An overview of (a) in-kernel attack design proposed in this work vs. (b) classical design in user space.

implement and optimize NitroSketch [19] with the goal of guaranteeing high packet rates. In our paper we implement in eBPF a different sketch specifically designed to detect and identify DDoS attacks [3], while taking into consideration most of the best practices identified in Ref. [7].

The same best practices of Ref. [7] have also been adopted by Ref. [20], which proposes a framework for runtime optimization of network code that includes an eBPF plugin. The framework adopts an auxiliary sketch implemented in eBPF to keep track of map accesses. Ref. [21] proposes a system for the automatic offloading of sketching operations to the Data Processing Unit when needed, showing that pure eBPF implementations may be inefficient. Our work is orthogonal to this work, as our goal is to perform sketch-based DDoS detection in kernel space, and our approach could further benefit by adopting frameworks as those of Refs. [20] [21].

Finally, we would like to stress that, to the best of our knowledge, the only paper adopting a sketch in kernel space for DDoS attack detection is Ref. [15], where a Bloom Filter is adopted. However, as already said, such a Bloom Filter is updated by a user space program, while in our case both *Query* and *Update* logic are fully performed in kernel space.

IV. Proposed Design

A. Design overview

When it comes to attack detection in the end hosts, a primary concern is the maintenance cost due to the hosts' distributed nature. As said, recent advancements of eBPF offer a non-disruptive way to deploy and modify functions at runtime, eliminating the need for recompilation. In this work, we leverage these advantages and propose an in-kernel double-sketch design that enables fast and memory-efficient attack detection within the kernel space of the end hosts.

An overview of our in-kernel sketch-based attack detection design is depicted in Fig. 1(a). Our strategy is mainly run in the kernel space and involves interaction with the user space within the host machine. Compared with the classical design where the detection services are run purely in the user space (as depicted in Fig. 1(b)), the key difference of our proposal is that the detection is done *directly in kernel space* upon traffic arrival, to avoid the latency caused by forwarding the traffic to the detection application running in the user space. Particularly, our design mainly involves two components: in-kernel *double-sketch-based detection* at runtime, and *self-adaptive threshold updates* assisted by the user space:

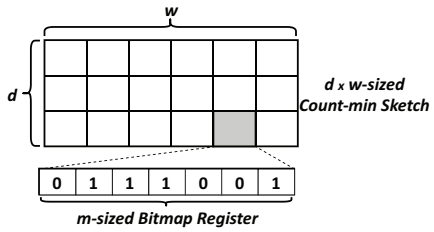


Fig. 2. Data structure of BACON Sketch.

1) *Double-sketch-based detection*: Upon traffic arrival, the traffic goes through an inspection based on our double-sketch design. The proposed double-sketch design can ensure the inspection of *every* flow, avoiding that any group of packets and/or flows arriving while the threshold is updated is skipped. Thanks to our design, the accuracy performance can be improved, at the expense of doubling memory occupation. The double-sketch-based detection service is written in eBPF for *seamless in-kernel configuration at runtime*. By exploiting eBPF's transparency, our double-sketch-based detection program can be loaded to the kernel without affecting the already-running network functions.
2) *Self-adaptive threshold updates*: To mitigate the limitations of static thresholds in classical sketch-based solutions over the dynamic nature of traffic flows, the design proposed in this work introduces threshold runtime updates in eBPF programs to enable an adaptive attack detection for better accuracy performance.

In the following, we report the main encountered challenges and implementation details of our proposed solutions related to our design. Our developed source code is available online[1].

B. In-kernel sketch for fast detection

Challenge: Existing solutions for DDoS attack detection in end hosts face challenges like high memory usage and computational overhead due to the massive volume of network traffic: this can cause degraded system performance and slower response times. Conventionally, sketch-based attack detection operates in user space, leading to an extra forwarding process from the kernel space up to the user space. In contrast, running sketch-based detection in kernel space via eBPF unlocks faster detection, as data can be processed at a lower system level, without incurring latency caused by the interaction with the user space. However, in-kernel detection poses challenges in complex development and maintenance due to the limitations faced in the kernel space. Specifically, the challenges are: *(i)* eBPF programs are restricted for safety and stability, i.e., to prevent that the loaded program can crash or corrupt the running kernel; *(ii)* the memory available for sketch computing is limited due to the constraints of eBPF maps; *(iii)* limitations in the computational complexity of eBPF programs are introduced to prevent negative impact on system performance.

Our solution: To efficiently sample and analyze large data streams in real-time and accommodate the eBPF constraints in the kernel space, a memory-efficient sketch data structure, the BACON (i.e., *BitmAp COuNt-min*) Sketch [3], is considered for in-kernel DDoS detection. BACON can reduce resource

[1]https://gitlab.com/federicodeiaco1/bacon_sketch

Algorithm 1 In-Kernel Packet Processing Algorithm

Initialization:
BACON Sketch ← All bits set to 0
E_{dsts} ← All values set to 0
$Threshold$ ← MAX_UINT32
while true **do**
  user space timer starts:
  while reading packet stream **do**
    Extract k_{src} and k_{dst} from packet header
    E_{dst} ← Execute BACON Sketch's *Update* and *Query* with k_{src} and k_{dst}
    $index$ ← Hash(k_{dst})
    Store E_{dst} in E_{dsts} at $index$
    if $E_{dst} > Threshold$ **then**
      Drop packet, it belongs to DDoS attack
    end if
  end while
  user space timer expires:
  Calculate new $Threshold$ in user space
  Reset BACON Sketch & all values
end while

consumption and enable quick detection of DDoS attacks without compromising system stability and efficiency.

Fig. 2 depicts the probabilistic data structure of BACON Sketch. It is designed to estimate the cardinality of flows directed towards a specific destination IP address [3], i.e., E_{dst}. It combines two data structures, the Direct Bitmap and the Count-min Sketch (see Section II), and leverages the advantages of these structures to optimize the system performance specifically for volumetric DDoS attack detection. By combining these two structures, BACON Sketch is able to provide a compact estimation of network traffic data in a memory-efficient manner, being especially suitable to meet the constraints of an in-kernel design. Algorithm 1 shows the performed packet processing operations. k_{src} (k_{dst}) is the source (destination) flow key, which may at least include the source (destination) IP address. A host can handle various destination flow keys, e.g. associated to different hosted VMs.

C. Double BACON Sketch for operation optimization

Challenge: Running BACON Sketch via eBPF in kernel space brings challenges in accurate detection due to eBPF's constraints. As shown in Algorithm 1, the BACON Sketch detection process involves two sequential phases for each packet: *(i) Update* phase, which increments array counters for new elements based on hash outputs, and *(ii) Query* phase, which estimates results by combining counter values from array positions determined by hash functions. Performing both phases in kernel space can lead to periods where some network traffic is skipped, as they are executed serially upon traffic arrival. This periodically results in packets not being processed for *n* seconds, where *n* is the duration of the threshold updating period. The issue is caused by modifications in shared memory structures during the *Update* phase, which is designed to ensure thread safety with atomic operations but can lead to delays, particularly under high traffic throughput.

Our solution: To address the aforementioned challenge, we proposed the adoption of a *double BACON Sketch* system to tackle eBPF's constraints and expedite sketch operations in the kernel space. Fig. 3 demonstrates the proposed design:

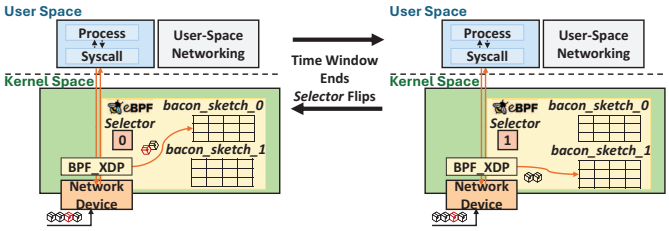


Fig. 3. Proposed design of double BACON Sketch.

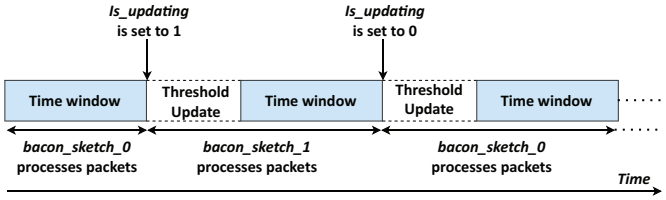


Fig. 4. Time window of traffic flow processing in double BACON Sketch.

two BACON Sketches are introduced to the detection process, named *bacon_sketch_0* and *bacon_sketch_1*. By doubling the sketches, race conditions between kernel and user space are mitigated and packet skipping is eliminated; thereby, the detection process is sped up and all packets can be inspected.

A selector (i.e., *is_updating*) is also introduced with binary values $\{0, 1\}$. The selector is initialized with the value 0, indicating that, during the first time window, all the *Update* and *Query* operations are performed on the *bacon_sketch_0* for each packet. When the time window reaches an end, the selector is switched to 1. This triggers the *Update* and *Query* operations to be carried out on the other sketch (i.e., *bacon_sketch_1*) for the subsequent traffic, while the user space handles the calculation of the new threshold and resets the involved data structures. This cycle continues, with the selector switching between 0 and 1 at the end of each time window, as illustrated in Fig. 4.

Auxiliary sketches for additional optimization: Counting the number of 1s inside the Bitmap registers for each BACON Sketch row causes a bottleneck in the *Query* function. Therefore, two auxiliaries Count-min Sketches have been introduced. Fig. 5 presents an idea of how they are employed during the packet processing to estimate E_{dst}. Essentially, each Bitmap register value that changes from 0 to 1 in a BACON Sketch will cause an increase of the corresponding counter in the associated Count-min Sketch. As can be noted from the figure, it is possible that the counters associated with a particular destination may vary across rows of the Count-min Sketch. This variability arises due to potential collisions in the hash function used to compute the indexes with different flow information: by querying the Count-min Sketch, the minimum value is retrieved and E_{dst} is estimated, thus minimizing the risk of overestimation.

D. Self-adaptive thresholds for dynamic traffic

Challenge: The BACON Sketch is based on a threshold to determine whether the arriving traffic flow is malicious. Conventional static thresholds [3] cannot adapt to the dynamic nature of network traffic, resulting in inaccurate detection.

Our solution: To improve the detection performance and adapt to the traffic dynamics, an adaptive threshold-based

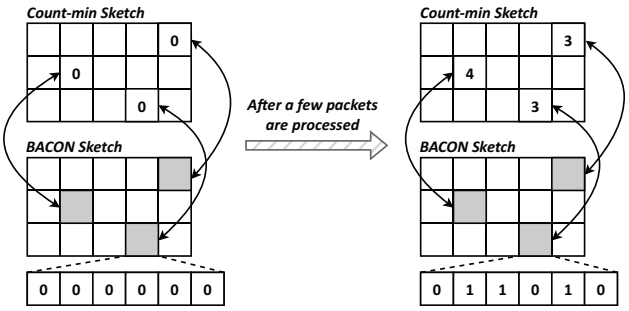


Fig. 5. Adoption of auxiliary Count-min Sketches.

detection method is proposed. We introduce a method based on Exponential Weighted Moving Average (EWMA) to compute dynamic thresholds by considering that current data is more relevant than the past one. EWMA can smooth out data in time series analysis by giving more weight to recent data samples and less weight to older ones. In the user space, the average $Ave_{E_{dst}}$ and deviation $Dev_{E_{dst}}$ of E_{dst} for each dst are continuously updated as in Eqs. 1 and 2, where $Curr_{E_{dst}}$ is the last recorded E_{dst}, retrieved from the eBPF maps (E_{dsts}). α and β control the sensitivity of EWMA to traffic changes.

$$Ave_{E_{dst}} = (1-\alpha)\cdot Ave_{E_{dst}} + \alpha \cdot Curr_{E_{dst}} \quad (1)$$

$$Dev_{E_{dst}} = (1-\beta)\cdot Dev_{E_{dst}} + \beta \cdot |Curr_{E_{dst}} - Ave_{E_{dst}}| \quad (2)$$

The user space then calculates the per-destination threshold as $Th_{dst} = Ave_{E_{dst}} + k \cdot Dev_{E_{dst}}$, with k called *margin*. The DDoS detection threshold is then an arithmetic mean of all the Th_{dst}, i.e., $Threshold = \frac{\sum_{dst} Th_{dst}}{N}$, where N is the number of destinations dst contacted by at least one source. $Threshold$ sets a dynamic boundary that adapts to benign traffic variations, as it is always updated at the end of each time window if no malicious behavior is detected. If E_{dst} exceeds $Threshold$, this indicates an anomalous traffic spike or potential network issues. Thereby, the system can automatically discern malicious packets from benign ones and adapt itself based on the type of detected ingress traffic.

V. Experimental Setup

For performance evaluation, experiments were run (unless otherwise specified) on a host machine with AMD Ryzen 5 5600X 12-core CPU @ 3.7GHz and 16GB DDR4 memory, configured with Ubuntu 22.04.4 LTS x86_64 with Linux kernel version 6.5.0-25-generic. The proposed sketch prototype is coded in C with eBPF library. The attack scenario includes many malicious hosts and a victim host, where attacker hosts generate malicious traffic to the victim host. The scenario has been emulated using a DDoS simulation tool[2] running on a single attacker host, generating packets with varying malicious IP addresses. The malicious traffic was captured and then replayed to the victim together with background traffic. The generated and used traces are available online[3].

Metrics: Multiple scores were calculated to assess detection results. Here, TP, FN, FP, TN indicate the number of True Positives, False Negatives, False Positives, and True Negatives. *Precision* (i.e., $\frac{TP}{TP+FP}$) presents the confidence

[2]https://github.com/ricardojoserf/ddos_simulation/tree/master
[3]https://dx.doi.org/10.5281/zenodo.14049439

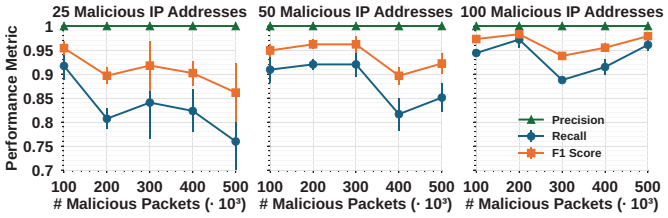


Fig. 6. Performance while varying the number of malicious IP addresses and of malicious packets per IP address.

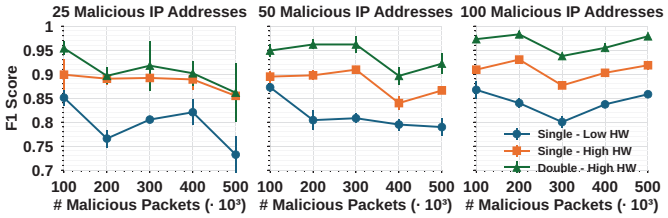


Fig. 7. Comparison between *double* BACON Sketch and *single* BACON Sketch, executed on *high*-performing hardware and *low*-performing hardware.

level for samples identified as benign. *Recall* (i.e., $\frac{TP}{TP+FN}$) shows the number of positive cases (i.e., DDoS attacks) correctly identified. *F1 Score* (i.e., $2\frac{Precision \cdot Recall}{Precision+Recall}$) is a harmonic mean of Precision and Recall.

Sketches configuration: We considered sketches with $(m = 2048) \times (w = 2048) \times (d = 3)$ ($\sim$ 25 MB of occupied memory when a *double sketch* is adopted). We also performed tests with smaller sketch sizes: $1024 \times 1024 \times 3$ ($\sim$ 6 MB), $1024 \times 2048 \times 1$ ($\sim$ 4 MB), $1024 \times 2048 \times 1$ ($\sim$ 4 MB) and obtained similar results. However, we focus on the first configuration as it better protects from traffic fluctuations.

VI. Performance Evaluation

A. Detection performance with increased malicious traffic

Fig. 6 presents the detection performance results on traffic with varying malicious IP addresses and number of malicious packets. The number of malicious packets is intended for each malicious IP address. Overall, the proposed design performs satisfactorily and robustly across different scenarios. Precision consistently remains high, indicating the model's ability to correctly identify malicious packets with minimal false positives. As the number of malicious IP addresses increases, F1 Score remains relatively stable. Not surprisingly, a higher number of malicious IP addresses makes it easier to detect the attack.

Fig. 7 shows two interesting behaviors. First, a *single sketch* implementation is tested on the host machine specified in Section V, called high-performing hardware (*High HW*), and on a AMD Ryzen 5 2500U 8-core @ 2.0GHz and 8GB DDR4 memory, called low-performing hardware (*Low HW*). It can be seen how the performance on *Low HW* is consistently worse than on *High HW*. The prolonged duration of the update threshold and reset phases results in discarding more packets, leading to increased performance degradation. This behavior is mitigated by adopting a *double sketch*. In this case the F1 Score is higher, as no packet is discarded during the update threshold and reset phases. In addition, the double-sketch system is more robust, since the same results obtained for *High HW* are also obtained for *Low HW* (not shown in the figure), at the expense of doubling the memory occupation.

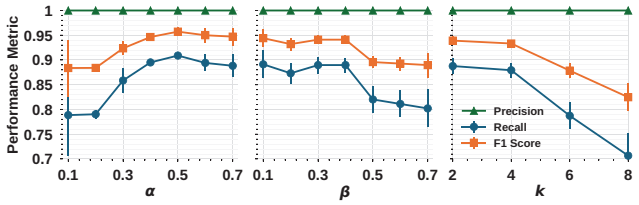


Fig. 8. Sensitivity evaluation of the dynamic threshold by varying α, β, k.

TABLE II
COMPARISON OF DETECTION TIME IN USER SPACE AND KERNEL SPACE

Strategy	Mean	Standard Deviation
BACON Sketch (kernel space)	0.52 μs	5.67 μs
BACON Sketch (user space)	4.2 ms	2.55 s

B. Sensitivity analysis to dynamic threshold parameters

Fig. 8 illustrates the sensitivity evaluation concerning the parameters α, β and k in the self-adaptive threshold design. The first two plots illustrate the EWMA's sensitivity to the estimated values. A low α prioritizes past values, while a high α emphasizes recent data. Performance generally improves up to $\alpha = 0.5$, then it slightly decreases. This indicates that α should be set slightly higher than 0.5 for optimal performance. Similar to α, a higher β increases the influence of recent deviations. A low β values past deviations more, and increasing β can degrade performance, highlighting the importance of historical deviation data. In this case, β should be set close to 0.1. Finally, as the margin k increases, performance shows a decline. This is because the $Threshold$ value gets higher, making the task of finding attacks associated to a lower number of flows harder. Such result indicates the selection preference on low k values, which would lead to better performance and adaptation to traffic dynamics.

C. System performance

Table II compares the processing latency of the in-kernel eBPF-based double sketch proposed in this work with the same sketch implemented in the user space[4] [3]. The proposed in-kernel sketch experiences a better latency performance, as packets are processed significantly earlier, before reaching the user space environment. Another advantage lies in the capability of the eBPF implementation to make real-time decisions on packets, including the ability to discard them or allow them to pass through to the user space. In contrast, the user space implementation lacks the ability to directly discard packets. Consequently, the utilization of eBPF technology affords a higher degree of control over network traffic.

VII. CONCLUSION

In this paper we introduced a sketching algorithm to enable in-kernel DDoS attack detection for fast response on hosts. We prototyped our design in eBPF, and proposed a double BACON Sketch and an adaptive threshold update mechanism as key optimizations to tackle the eBPF limitations and boost detection performance. Our results show that the proposed solution is effective in providing good detection performance (F1 Score higher than 90%) and ensures orders-of-magnitude faster detection compared to an existing user-space design. Future work will be focused on exploring advanced algorithms (such as *HyperLogLog*) to enable the accurate detection of more types of attacks and the filtering of their related packets.

[4]https://gitlab.com/federicodeiaco1/bacon_sketch_us

ACKNOWLEDGMENT

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

REFERENCES

[1] Q. Li, H. Huang, R. Li *et al.*, "A Comprehensive Survey on DDoS Defense Systems: New Trends and Challenges," *Computer Networks*, 2023.
[2] F. Golpayegani, N. Chen, N. Afraz *et al.*, "Adaptation in Edge Computing: A Review on Design Principles and Research Challenges," *ACM Transactions on Autonomous and Adaptive Systems*, 2024.
[3] D. Ding, M. Savi, F. Pederzolli *et al.*, "In-network Volumetric DDoS Victim Identification using Programmable Commodity Switches," *IEEE Transactions on Network and Service Management*, 2021.
[4] Z. Tian, W. Shi, Y. Wang *et al.*, "Real-Time Lateral Movement Detection Based on Evidence Reasoning Network for Edge Computing Environment," *IEEE Transactions on Industrial Informatics*, 2019.
[5] F. Parola, R. Procopio, R. Querio *et al.*, "Comparing User Space and In-kernel Packet Processing for Edge Data Centers," *ACM SIGCOMM Computer Communication Review*, 2023.
[6] M. A. Vieira, M. S. Castanho, R. D. Pacífico *et al.*, "Fast Packet Processing with eBPF and XDP: Concepts, Code, Challenges, and Applications," *ACM Computing Surveys*, 2020.
[7] S. Miano, X. Chen, R. Ben-Basat *et al.*, "Fast In-kernel Traffic Sketching in eBPF," *ACM SIGCOMM Computer Communication Review*, vol. 53, no. 1, pp. 3–13, 2023.
[8] A. Sadiq, H. J. Syed, A. A. Ansari *et al.*, "Detection of Denial of Service Attack in Cloud Based Kubernetes Using eBPF," *Applied Sciences*, vol. 13, no. 8, p. 4700, 2023.
[9] H. J. Hadi, M. Adnan, Y. Cao *et al.*, "iKern: Advanced Intrusion Detection and Prevention at the Kernel Level using eBPF," *Technologies*, vol. 12, no. 8, p. 122, 2024.
[10] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann *et al.*, "The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel," in *ACM CoNEXT*, 2018.
[11] C. Estan, G. Varghese, and M. Fisk, "Bitmap Algorithms for Counting Active Flows on High Speed Links," in *ACM IMC*, 2003.
[12] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: the Count-min Sketch and its Applications," *Journal of Algorithms*, 2005.
[13] X. Zhang, L. Chen, and J. Bai, "SYN Flood Attack Detection and Defense Method Based on Extended Berkeley Packet Filter," in *ICNC-FSKD*, 2022.
[14] S. Miano, R. Doriguzzi-Corin, F. Risso *et al.*, "Introducing SmartNICs in Server-based Data Plane Processing: the DDoS Mitigation Use Case," *IEEE Access*, vol. 7, pp. 107 161–107 170, 2019.
[15] N. Kostopoulos, D. Kalogeras, and V. Maglaris, "Leveraging on the XDP Framework for the Efficient Mitigation of Water Torture Attacks within Authoritative DNS Servers," in *IEEE NetSoft*, 2020.
[16] S. A. Atiiq and C. Gehrmann, "X-Pro: Distributed XDP Proxies against Botnets of Things," in *NordSec*, 2021.
[17] A. Feraudo, D. A. Popescu, P. Yadav *et al.*, "Mitigating IoT Botnet DDoS Attacks through MUD and eBPF based Traffic Filtering," in *ICDCN*, 2024.
[18] M. Abranches, O. Michel, E. Keller *et al.*, "Efficient Network Monitoring Applications in the Kernel with eBPF and XDP," in *IEEE NFV-SDN*, 2021.
[19] Z. Liu, R. Ben-Basat, G. Einziger *et al.*, "Nitrosketch: Robust and General Sketch-based Monitoring in Software Switches," in *ACM SIGCOMM*, 2019.
[20] S. Miano, A. Sanaee, F. Risso *et al.*, "Morpheus: A Run Time Compiler and Optimizer for Software Data Planes," *IEEE/ACM Transactions on Networking*, 2024.
[21] X. Chen, X. Sun, W. Zhang *et al.*, "Accelerating Sketch-based End-Host Traffic Measurement with Automatic DPU Offloading," in *IEEE INFOCOM*, 2024.