INDDoS+: Secure DDoS Detection Mechanism in Programmable Switches

Damu Ding[1], Ozlem Kesgin[2], and Noa Zilberman[1]

[1]University of Oxford, Oxford, United Kingdom
[2]University of Edinburgh, Edinburgh, United Kingdom
{*damu.ding, noa.zilberman*}*@eng.ox.ac.uk*, z.o.kesgin@sms.ed.ac.uk

***Abstract*—Volumetric distributed Denial-of-Service (DDoS) attack is a key issue in modern telecommunication networks since it can exhaust the resources of legitimate users and cripple network services. Recently, with the emergence of high-throughput and low-latency programmable switches, DDoS detection mechanisms have been designed and implemented in an in-network manner, that is, DDoS detection executed directly within programmable switches. State-of-the-art works use advanced data structures to monitor the number of connections targeting destination hosts: if there is sudden increase of connections and the number exceeds a given threshold, the destination host is most likely under DDoS attack. However, while this approach is efficient in DDoS victims identification, it has inherent vulnerabilities in the detection mechanism that may lead to security issues. In this paper, we study two possible vulnerabilities in DDoS detection data structures, showing the possibilities to break DDoS detection mechanisms in programmable switches. To mitigate the constructed attacks, we propose a solution called INDDoS+. The results show that INDDoS+ is robust to attacks and can accurately detect DDoS attempts when limited hardware resources are assigned.**

I. Introduction

Volumetric distributed Denial-of-Service (DDoS) attack is becoming a major concern in computer networks. It sends large amounts of traffic from multiple compromised hosts to overwhelm the network resources of victims. To detect such an attack, network operators need to frequently collect network traffic and infer network status statistics [1], [2]. Nonetheless, modern intrusion detection systems face challenges when it comes to assessing network data streams, primarily due to the widespread distribution and sheer volume of network traffic. As programmable switches can provide high throughput and low latency, volumetric DDoS detection starts to be deployed directly on the switches. Due to limited hardware resources in the switch, collecting exact measurements of network traffic is impractical. Therefore, using probabilistic data structures, called *Sketches*, is a viable solution. Sketches require only a very small amount of memory but can guarantee high accuracy in the estimation of network measurements.

As shown in Fig.1, in recent years, sketches are offloaded to programmable switches for monitoring purpose (also called *in-network monitoring*), with the goal to maximize the computing speed for high performance monitoring. For example, a prominent work called INDDoS (In-network DDoS detection) [3], employs a data structure called BACON Sketch to monitor the number of distinct source IPs (equivalent to the number of connections or requests) targeting different destination hosts in the network. The switch is deployed at the edge of the network, so that it can track the overall network status. When

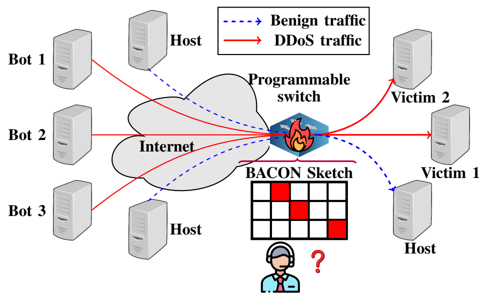

Fig. 1. Threat model

the number of connections to a host unexpectedly increases and surpasses the pre-defined threshold, the destination host is considered as a DDoS victim.

Even though DDoS detection mechanisms such as INDDoS have high accuracy in DDoS victim identification, they have no protection mechanisms. Attackers can attempt to mislead the Sketches with crafted traffic (e.g. spoofed source IPs), which may cause large bias on the estimation results of connections. Inaccurate estimations may further lead to wrong decisions on DDoS victim identification or leave DDoS attacks undetected.

To mitigate the problem, this paper investigates the vulnerabilities of sketch-based DDoS detection mechanisms, with BACON Sketch in INDDoS as an example. We show that attackers can either evade the DDoS detection of INDDoS or trigger false DDoS alarms by exploiting security issues in the Sketch. For example, attackers can manipulate the source IPs to avoid the increments of the estimated number of connections. Likewise, attackers can also inflate the Sketch with spoofed source IPs to cause detection false positives. With those threats in mind, we propose INDDoS+, a new solution that effectively mitigates security issues in INDDoS while providing good DDoS detection performance given similar hardware resources.

In summary, we make the following contributions:

- We present two different potential threats on sketch-based DDoS detection mechanisms and explain how to construct corresponding attacks.
- We design and develop a novel approach called INDDoS+ to detect and mitigate the threats on BACON Sketch.
- We evaluate our approach using real-world Internet traces, and discuss the trade-off between threats detection's accuracy and hardware constraints.

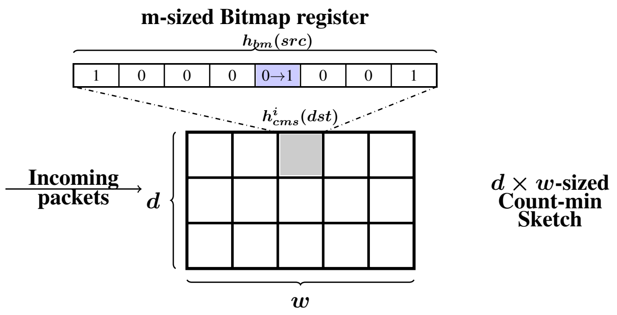

Fig. 2. The design of BACON Sketch

Algorithm 1: BACON Sketch

```
Input: Packet stream $S$ identified by $src$ or $dst$
Output: An estimated number of connections $\hat{E}_{dst}$ contacting $dst$
1  $d \leftarrow$ Number of hash functions in Count-min
2  $w \leftarrow$ Output size of hash functions in Count-min
3  $m \leftarrow$ Bitmap-register size
4  $\mathcal{B} \leftarrow$ A $d \times w \times m$-sized BACON Sketch
5  Function Update($src$, $dst$, $h_{cms}$, $h_{bm}$):
6      for Each hash function $h^i_{cms}$ in row $i$ do
7          $idx \leftarrow (h^i_{cms}(dst)\%w) \cdot m + h_{bm}(src)\%m$
8          if $\mathcal{B}_i[idx]$ is 0 then
9              $\mathcal{B}_i[idx] \leftarrow 1$
10 Function Query($dst$, $h_{cms}$, $h_{bm}$):
11     $\hat{E}_{dst} \leftarrow 0$
12     for Each hash function $h^i_{cm}$ in row $i$ do
13         $start \leftarrow (h^i_{cms}(dst)\%w) \cdot m$
14         $E_i \leftarrow \sum_{j=start}^{start+m-1} \mathcal{B}_i[j]$
15     $\hat{E}_{dst} \leftarrow \min(E_i)$ return $\hat{E}_{dst}$
16 Function Reset():
17     for Each row $i$ ($i \in [0 : d-1]$) do
18         $\mathcal{B}_i[0 : w \times m - 1] \leftarrow 0$
```

II. Background of BACON Sketch

In this work we focus on the recent BACON Sketch [3], which is a memory-efficient data structure used to estimate the number of connections (e.g., source hosts) targeting the same destination host.It has been recently used to detect DDoS attacks in programmable switches.

A. Basic operation of BACON Sketch

As shown in Fig.2, BACON Sketch is composed of a $d \times w$-sized Count-min Sketch [4] and an m-sized Bitmap [5] register: Count-min Sketch is a probabilistic data structure to estimate the number of packets in each flow, whereas Bitmap is a simple data structure used to count the number of distinct flows. By combining them, it is possible to count the number of distinct flows towards multiple destinations: in each bucket of Count-min Sketch, the Bitmap register is used to count the number of distinct flows. Therefore, the overall size of BACON Sketch is $d \times w \times m$. Algorithm 1 illustrates the details of BACON Sketch, which applies three actions: *Update*, *Query*, and *Reset*.

Update: Each row i ($0 \leq i \leq d-1$) is associated with a hash function h^i_{cms}, and all hash functions (e.g., CRC32 [6], md5 [7]) are pairwise independent. For all incoming packets, their destination IPs dst are hashed to the bucket $h^i_{cms}(dst)\%w$ ($0 \leq i \leq d-1, 0 \leq j \leq w-1$) in each row i ($0 \leq i \leq d-1$). Then the source IPs src are hashed by another hash function h_{bm} to determine the position $h_{bm}(src)\%m$ in the Bitmap. Therefore, the index idx to access the value is $idx = (h^i_{cms}(dst)\%w) \cdot m + h_{bm}(src)\%m$ (Line 7). If the value in the position idx is 0, then the value is updated to 1 (Lines 8-9). Otherwise, the value does not vary and is kept at 1.

Query: Lines 12-15 in Algorithm 1 show how *Query* in BACON Sketch works: during the time interval, for any dst, the sum E_i of all values in the Bitmap of row i (i.e., values in the range $[h^i_{cms}(dst) \cdot m : h^i_{cms}(dst) \cdot m + m - 1]$) represents the number of connections estimated by the row. Finally, the minimum E_i among all d rows (i.e., E_i with the smallerst number of collisions in the sketch), denoted by $\hat{E}_{dst}$, is the estimated real-time number of connections targeting the destination host dst.

Reset: At the end of each time interval, similar to other general sketches, all counters within the BACON Sketch are reset to 0 for the monitoring in the next time interval, as illustrated in Lines 17-18.

B. DDoS detection using BACON Sketch

INDDoS [3] uses a BACON Sketch in a P4 programmable switch ASIC (e.g. Intel Tofino switch) [8] to identify DDoS victims. The switch is deployed at the edge of the network to monitor the number of connections to different destination hosts. When there is a volumetric DDoS attack in the network, a number of compromised hosts (e.g. bots) attempt to send requests for connection with the DDoS victims and exhaust their available resource. Therefore, if the number of connections estimated by the Sketch suddenly increases and exceeds a given threshold, then the destination host can be identified as a DDoS victim.

III. Threat models on the Sketch

Even though INDDoS has high accuracy in DDoS victim identification, the BACON Sketch itself is vulnerable: attackers may tamper with values in the sketch by sending customized traffic, aiming to trigger false DDoS alarms or evade detection. In this paper, we investigate two possible threats on the BACON Sketch: inflating attack and evasion attack. Note that similar threats can also be applied to other types of sketches for the estimation of distinct connections, such as Multiresolution Bitmap [5], PCSA [9] and HyperLogLog [10].

We consider the case where the BACON Sketch is a black box, that is, the attacker does not know the hash functions used by the Sketch. The hash functions are not typical hash functions (i.e., the parameters in hash functions are well configured) and cannot be easily reversed. Nevertheless, the attacker can still access a shadow copy of the sketch via its APIs. A potential scenario can be that the attacker can install a well-encrypted P4 program into its own programmable switch. For BACON Sketch, only the default APIs (*Update*, *Query*, and *Reset*) are allowed. Therefore, it is not possible to decrement the counter values. The Sketch size $d \times w \times m$ is defined in the P4 program, but these parameters are invisible

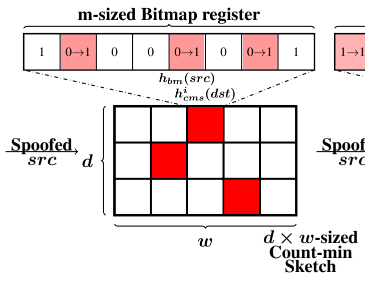

Fig. 3. Inflating attacks

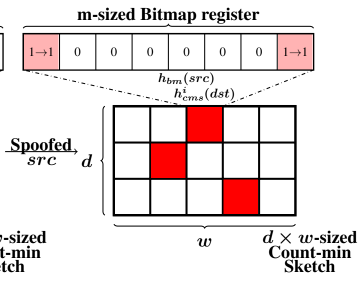

Fig. 4. Evasion attacks

to the attacker. The attacker can capture the benign traffic and populate it in the shadow copy to construct attack vectors. This attack scenario has been proved feasible for HLL sketches as described in [11] and [12]. Given the packet rate of programmable switches exceeds billions of packets per second, even though the monitoring interval is only in the order of seconds, it is still possible for attackers to insert the attack vector within a short amount of time.

A. Inflating attack

The goal of an inflating attack is to increase the number of estimated connections to a perceived victim. As shown in Fig.3, an attacker inserts a sequence of packets into the sketch with spoofed source IP src. The hashed flows identified by src flip the Bitmap entries from 0 to 1, significantly increasing the estimated connections to destination hosts, and leading to false positives in DDoS victim identification.

Algorithm.2 shows how to generate inflating attack traffic: initially, as shown in Lines 3-4, the attackers replay the captured benign traffic S within a given time interval to increment the counters in the BACON Sketch $\mathcal{B}$. Given a targeting victim dst^{target}, the attackers can query the number of source IPs towards dst^{target}, denoted by $\hat{E}^{org}_{dst}$. Then they add a source IP src^{target} together with the victim dst^{target} into the Sketch. The source IPs src^{target} in the synthetic set $\mathcal{A}$ can be generated as a 32-bits number and converted to an IP address. The range of $\mathcal{A}$ can be customized since some IP addresses are invalid or private, and as the IP ranges of local service providers are more likely. The number of source IPs to the victim dst^{target}, called $\hat{E}_{dst}$, is queried again (Lines 6-8). If $\hat{E}_{dst}$ is greater than $\hat{E}^{org}_{dst}$, then this source IP src^{target} is added to the attack set $\mathcal{F}$ (Lines 9-10). Once $\hat{E}_{dst}$ is fully filled and reaches the largest size of Bitmap (i.e., m), the attack construction is stopped (Lines 11-12). Finally, the attack set $\mathcal{F}$ is constructed and available to launch inflating attacks.

B. Evasion attack

Fig.4 shows how an attacker can send DDoS traffic to the victim while evading detection by INDDoS using BACON Sketch. The attacker has a set of flows with spoofed source IPs src coming from bots, and wants the flows not to increment the estimated connection number of BACON Sketch. This can be achieved if the targeted sketch slot $(h^i_{cms}(dst)\%w) \cdot m + h_{bm}(src)\%m$ is already 1. This way, the estimated number of connections targeting the destination hosts does not vary, and the victims under DDoS attacks will not be reported to the network operator.

Algorithm 2: Inflating attack construction

```
Input: A non-DDoS-victim destination host with IP $dst^{target}$, a synthetic set $\mathcal{A}$ of spoofed IPs
       $src^{attack}$, and a packet stream $S$ in a given time interval $T_{int}$
Output: A set $\mathcal{F}$ with different spoofed IPs $src^{attack}$ that makes $dst^{target}$ be a DDoS victim
1  $\mathcal{B} \leftarrow d \times w \times m$-sized BACON Sketch with hash function $h_{cms}$ in Count-min and $h_{bm}$ in Bitmap
2  $\mathcal{F} \rightarrow \{\}$
3  for Each packet in S (with src and dst) do
4      $\mathcal{B}$.Update($src$, $dst$, $h_{cms}$, $h_{bm}$)
5  for Each source IP $src^{attack}$ in $\mathcal{A}$ do
6      $\hat{E}^{org}_{dst} \leftarrow \mathcal{B}$.Query($dst^{target}$, $h_{cms}$, $h_{bm}$)
7      $\mathcal{B}$.Update($src^{attack}$, $dst^{target}$, $h_{cms}$, $h_{bm}$)
8      $\hat{E}_{dst} \leftarrow \mathcal{B}$.Query($dst^{target}$, $h_{cms}$, $h_{bm}$)
9      if $\hat{E}_{dst} > \hat{E}^{org}_{dst}$ then
10         $\mathcal{F}$.add($src^{attack}$)
11     if $\hat{E}_{dst} == m$ then
12         return $\mathcal{F}$
```

Algorithm 3: Evasion attack construction

```
Input: A targeted DDoS victim host with IP $dst^{target}$ and a synthetic set $\mathcal{A}$ of spoofed IPs
       $src^{attack}$ , and a packet stream $S$ in a given time interval $T_{int}$
Output: A list $\mathcal{L}$ containing different spoofed IPs $src^{attack}$ that avoid $dst^{target}$ to be identified
        as a DDoS victim
1  $B \leftarrow d \times w \times m$-sized BACON Sketch with hash function $h_{cms}$ in Count-min and $h_{bm}$ in Bitmap
2  $\mathcal{L} \rightarrow \{\}$
3  for Each source IP $src^{attack}$ in $\mathcal{A}$ do
4      for Each packet in S (with src and dst) do
5          $\mathcal{B}$.Update($src$, $dst$, $h_{cms}$, $h_{bm}$)
6      $\hat{E}^{org}_{dst} \leftarrow \mathcal{B}$.Query($dst^{target}$, $h_{cms}$, $h_{bm}$)
7      $\mathcal{B}$.Update($src^{attack}$, $dst^{target}$, $h_{cms}$, $h_{bm}$)
8      $\hat{E}_{dst} \leftarrow \mathcal{B}$.Query($dst^{target}$, $h_{cms}$, $h_{bm}$)
9      if $\hat{E}^{org}_{dst} == \hat{E}_{dst}$ then
10         $\mathcal{L}$.add($src^{attack}$)
11     $\mathcal{B}$.Reset()
12 return $\mathcal{L}$
```

As shown in Algorithm.3, similar to Inflating attack construction, Lines 4 to 5 show that the attackers need to populate the captured benign traffic S to increment the counters in BACON Sketch $\mathcal{B}$ first. After querying the number of source IPs $\hat{E}^{org}_{dst}$ contacting the victim dst^{target}, a packet with $\{src^{attack}$, $dst^{target}\}$ is updated to the sketch. If the estimated number of source IPs to dst^{target}, called $\hat{E}_{dst}$, remains unchanged, then src^{attack} will be considered as a candidate of evasion attack and added to the list $\mathcal{L}$ (Lines 6-10). Since the Sketch does not allow to decrement the counters, the attackers must reset all counters in the sketch to 0 (Line 11) and repeat the same operation until all src^{attack} in $\mathcal{A}$ (i.e., a synthetic set of spoofed IPs) are tried. The attackers are required to reset the counters to reduce test time, otherwise they can wait for the counters reset at the end of every time interval. Afterwards, the attackers can spoof the source IP for packets in DDoS traffic using the elements in $\mathcal{L}$ to evade the DDoS detection mechanism of INDDoS.

IV. Secure in-network DDoS victim identification

In this section, we introduce INDDoS+, an enhanced in-network DDoS victim identification system using a BACON Sketch that can effectively detect evasion and inflating attacks.

INDDoS+ operation is introduced in Algorithm 4: there are two BACON Sketches, $\mathcal{B}1$ and $\mathcal{B}2$, deployed in the switch. The main difference between these two Sketches is that they use different hash functions for Bitmap, that is, h_{bm1} for $\mathcal{B}1$ and h_{bm2} for $\mathcal{B}2$. Since *(i.)* attackers are only able to spoof the source IPs src to mislead the output of BACON Sketch and *(ii.)* the hash functions in Count-min Sketch, namely h_{cms}, is only responsible for hashing the destination IP dst, changing the hash function h_{cms} is optional in INDDoS+. In our evaluation, the hash functions in h_{cms} remain the same for both. During each time interval, the two BACON Sketches $\mathcal{B}1$ and $\mathcal{B}2$ are updated using different hash functions for the Bitmap. Following updates, both of sketches can query the real-time number of hosts contacting the destination (Lines 5-8). If the queried values, $\hat{E}1_{dst}$ from $\mathcal{B}1$ and $\hat{E}2_{dst}$ from $\mathcal{B}2$, are greater than the given DDoS detection threshold Tr, it means that the destination host dst is likely under attack. The switch can report the victim IP dst to controller for further mitigation (Lines 9 - 10). However, if either $\hat{E}1_{dst}$ or $\hat{E}2_{dst}$ is larger than Tr, this indicates that *(i.)* attackers evade the increments in BACON Sketch or *(ii.)* Attackers inflate the values in BACON Sketch. In either case, the BACON Sketch is under attack, and the switch must report this $alarm$ to a controller (Lines 11 - 12). The controller needs to change the hash function used by Bitmap to protect BACON Sketch and avoid consequent attacks. This can be done by installing an updated P4 program into the switch.

Algorithm 4: INDDoS+

```
Input: Packet stream $S$ identified by $src$ or $dst$
Output: DDoS victim IP address $dst$ or $alarm$
        reporting INDDoS+ under attack
1  $Tr \leftarrow$ DDoS detection threshold
2  $\mathcal{B}1 \leftarrow d \times w \times m$-sized BACON Sketch with hash
     function $h_{cms}$ in Count-min and $h_{bm1}$ in Bitmap
3  $\mathcal{B}2 \leftarrow d \times w \times m$-sized BACON Sketch with hash
     function $h_{cms}$ in Count-min and $h_{bm2}$ in Bitmap
4  for Each packet in $S$ (with $src$ and $dst$) do
5      $\mathcal{B}1.Update(src, dst, h_{cms}, h_{bm1})$
6      $\hat{E}1_{dst} \leftarrow \mathcal{B}1.Query(dst, h_{cms}, h_{bm1})$
7      $\mathcal{B}2.Update(src, dst, h_{cms}, h_{bm2})$
8      $\hat{E}2_{dst} \leftarrow \mathcal{B}2.Query(dst, h_{cms}, h_{bm2})$
9      if $\hat{E}1_{dst} > Tr$ and $\hat{E}2_{dst} > Tr$ then
10         Report $dst$
11     else if $\hat{E}1_{dst} > Tr$ or $\hat{E}2_{dst} > Tr$ then
12         Report $alarm$
13 Reset()
```

V. Experimental evaluation

INDDoS+ was implemented in Python to study its performance in detecting attacks on BACON Sketch.

A. Evaluation metrics and settings

1) Testing flow traces: CAIDA 2018 [13] trace is used as benign traffic. For testing the two threats, we used the first 5 seconds of a trace, and tested with 5 different time intervals, ranging from 1s to 5s. To test the DDoS victim identification performance, we used a 50s flow trace, and split it into 10 time intervals with 5 seconds each. Each time interval contains nearly 2.3 million packets and 60 thousand unique source IPs.

2) Evaluation metrics: For testing inflating attacks, we consider the number of keys and the number of attempts required to inflate the BACON Sketch in INDDoS. For evasion attacks, we measure the number of vulnerable keys that can evade DDoS detection by INDDoS. To compare the DDoS victim identification performance of INDDoS and INDDoS+ using the same amount of memory, we considered Recall Re, Precision Pr, and F1 score $F1$ as the key metrics. Considering that *(i.)* True Positive (TP) is the number of DDoS victims that are correctly identified, *(ii.)* False Negative (FN) is the number of undetected DDoS victims, and *(iii.)* False Positive (FP) is the number of wrongly identified DDoS victims, the metrics introduced above are defined as follows:

$$Re = \frac{TP}{TP+FN} \quad Pr = \frac{TP}{TP+FP} \quad F1 = \frac{2 \cdot Re \cdot Pr}{Re + Pr}$$

3) Parameter configurations: The hash functions used in Count-min Sketch within BACON Sketch is *md5*: each dst is hashed with $md5$ and then hashed again with the row number i. The hash function used in the Bitmap of the first BACON Sketch is $md5$ as well, whereas that in the second BACON Sketch of INDDoS+ is also $md5$ but hashed second time with a parameter 100. We also considered another case called BACON Sketch+, where the hash functions of Bitmaps in BACON Sketch+ are different for each row: BACON Sketch+ hashes src in Bitmap with $md5$ and also the row number i. For instance, the Bitmap in the first row of BACON Sketch+ hashes src with the parameter 1. For the Bitmap at the second row, the parameter is 2. If not otherwise specified, the BACON Sketch and BACON Sketch+ size is the same as used in [3]: the Count-min size $d \times w = 3 \times 1024$, and the Bitmap size m is 1024. The DDoS detection threshold Tr is 0.5% of overall number of source IPs in each time interval, around 300.

B. Exp 1: Number of keys and attempts for inflating attacks

In this experiment, we consider two cases: *(i.)* the destination host dst with the smallest number of source hosts src contacted and *(ii.)* the destination host dst with the largest number of source hosts src contacted in a given time interval. We generate keys (i.e., source IPs) using $2^{16} - 1 = 65535$ consecutive source IPs in this experiment. When the duration of time interval increases, the threshold increases as well because the threshold depends on the proportion of the overall number of source IPs during time interval. As shown in Table.I, less than 100 keys and attempts are required to mislead

TABLE I
NUMBER OF KEYS AND ATTEMPTS REQUIRED FOR INFLATING ATTACKS

Time interval	Smallest $\#src$ to dst	Detection threshold (0.5% of total $\#src$)	#keys to exceed the threshold		#attempts to exceed the threshold		#keys to full fill the Bitmap		#attempts to exceed the threshold		Largest $\#src$ to dst	#keys to full fill the Bitmap		#attempts to exceed the threshold	
			BACON Sketch	BACON Sketch+	BACON Sketch	BACON Sketch+	BACON Sketch	BACON Sketch+	BACON Sketch	BACON Sketch+		BACON Sketch	BACON Sketch+	BACON Sketch	BACON Sketch+
1s	1	101	78	48	81	52	971	969	8513	9874	213	933	982	8513	9874
2s	1	176	49	49	59	52	892	888	8513	9874	309	881	890	8513	9874
3s	1	232	50	49	66	54	838	835	8513	9874	377	824	833	8513	9874
4s	1	279	56	46	81	52	798	789	8513	9874	439	782	789	8513	9874
5s	1	320	58	47	85	57	760	748	8513	9874	485	748	748	8513	9874

TABLE II
EVASION ATTACK: NUMBER OF VULNERABLE KEYS IN 64K SAMPLES

Time interval	Smallest $\#src$ to dst	#Vulnerable keys		Largest $\#src$ to dst	#Vulnerable keys	
		BACON Sketch	BACON Sketch+		BACON Sketch	BACON Sketch+
1s	1	3456	3507	213	4051	3963
2s	1	8564	8749	309	9077	8583
3s	1	12406	12480	377	12801	12216
4s	1	14607	15407	439	15530	15055
5s	1	17504	17870	485	17668	17701

TABLE III
COMPARISON OF DDOS VICTIM IDENTIFICATION PERFORMANCE BETWEEN INDDOS [3] AND INDDOS+

Strategy	**BACON Sketch size** ($d \times w \times m$)	**# BACON Sketches**	**Recall**	**Precision**	**F1 score**
INDDoS	$3 \times 1024 \times 1024$	1	0.96	0.99	0.97
INDDoS+	$3 \times 512 \times 1024$	2	0.96	0.79	0.86
INDDoS+	$3 \times 1024 \times 512$	2	0.17	0.96	0.28
INDDoS+	$3 \times 1024 \times 1024$	2	0.96	0.99	0.97

BACON Sketch to consider a legitimate host with the smallest number of connections as a DDoS victim. This number is even smaller if BACON Sketch+ is used because of the hash collisions caused by Bitmap. Another observation is that the number of keys required to fill the Bitmap to the indicated destination host decreases with the length of time interval. This is because a larger time interval has more packets, and there are more 1s in BACON Sketch. However, all of them need the same number of attempts, that is, 8513 packets. Instead, BACON Sketch+ requires relatively smaller number of keys to fill Bitmap because more collided 1s are generated by BACON Sketch+ when the number of source IPs is small. Since src are hashed to different positions of different rows, more attempts (i.e. packets with different spoofed src) are required for BACON Sketch+ to fully fill the Bitmap, that is, 9874 times. When transitioning to the dst with the largest number of source IPs (i.e., the destination host contacted by the most connections), since the number is already greater than the threshold, we only consider the number of keys to fill the Bitmap. The number of keys required is smaller than the destination hosts with the smallest number of connections for BACON Sketch, but the number of attempts remains the same. The reason is that the 8513th key is the last key that can flip the ultimate 0 in Bitmap targeting dst to 1. Likewise, BACON Sketch+ requires more attempts than BACON Sketch because the positions with 0s are more difficult to find to inflate when the number of src is large. Overall, in our tests, the attackers needed to send less than 10,000 packets to mislead INDDoS (either with BACON Sketch or BACON Sketch+) to label all destination hosts as a DDoS victim. This demonstrates that both Sketches are very vulnerable to this attack.

C. Exp 2: Number of keys for evasion attacks

Similar to Exp 1, as shown in Table.II, we generated 65535 source IPs with as a similar prefix. In this experiment, we would like to see how many vulnerable keys exist within 65535 samples. We used the CAIDA trace ranging from 1 second to 5 seconds as the benign traffic in BACON Sketch and then try to find possible vulnerable keys. Intuitively, the number of vulnerable keys increase as the time interval length increases because there are more packets in the BACON Sketch. Even though the destination host has only 1 source host connected, when the time interval increases from 1s to 5s, the vulnerable keys significantly increases from 3456 to 17504. For BACON Sketch+, due to the collisions caused by different hash functions in Bitmap when the number of src is small, there are usually more 1s in Bitmap, and the number of vulnerable keys is higher than that of BACON Sketch. On the other hand, the destination hosts with the largest number of contacting source IPs have much more vulnerable keys because there are more 1s in the BACON Sketch, and the attackers can generate packets to evade the increments with higher probability. However, in this case, BACON Sketch+ has a smaller number of vulnerable keys because it has less 1s when the number of connections is larger than the BACON Sketch. The results reveal that with only 65535 samples attempted, it is possible for attackers to find at least thousands of available vulnerable keys for the spoofed IPs of their attack traffic. Both BACON Sketch and BACON Sketch+ are vulnerable to the two aforementioned attacks, and changing the parameters inside the Sketch cannot completely mitigate them. Therefore, the comparison of INDDoS and INDDoS+ considers only BACON Sketch.

D. Exp 3: DDoS victim identification with equal resources

Since INDDoS+ needs two BACON Sketches to secure the DDoS victim identification mechanism of INDDoS, for a fair comparison we would like to see the identification performance if the same amounts of memory is assigned. As shown in Table.III, we used the default size of BACON Sketch (i.e. $d \times w \times m = 3 \times 1024 \times 1024$) in INDDoS. This is because the BACON Sketch with this size can perform the best F1 score on DDoS detection as demonstrated in [3]. To guarantee the same amount of memory is used, either the Bitmap size m or the output size of Count-min w should be halved. Therefore, we tested INDDoS+ with two different sizes: $d \times w \times m = 3 \times 512 \times 1024$ and $3 \times 1024 \times 512$. The results are the average of 10 time intervals. When $m = 512$, there are more collisions in Bitmap, so the estimated number of connections to a destination host is decreased. This means that there are more undetected DDoS victims, resulting in a recall of only 0.17. Instead, when we use $w = 512$, the largest sufficient number estimated by Bitmap is still 1024, so INDDoS+ is able to detect all possible DDoS victims. However, due to the collisions caused by the decreased size of w, the estimated number is increased: there are more wrongly identified victims, leading to lower precision, which is 0.79. In our opinion, the F1 score of INDDoS+ with size $d \times w \times m = 3 \times 512 \times 1024$ (i.e., 0.86) is acceptable when memory is limited. It depends on the choice of network operators: using a single larger-sized BACON Sketch within INDDoS may be under attack and lead the F1 score to a very small value but can provide a higher detection accuracy, whereas INDDoS+ containing two smaller-sized BACON Sketches has slightly lower precision but is robust to potential attacks. However, if sufficient memory is available in the hardware, INDDoS+ is the better choice as it can guarantee comparable detection performance as INDDoS.

VI. Discussion

The key limitation to implementing INDDoS+ on programmable switches, such as Intel Tofino switches, is not the memory but the number of *pipeline stages*. Since the P4 [8] implementation of BACON Sketch is identical for INDDoS and INDDoS+, given that INDDoS has utilized all available $stages$ in Intel Tofino switch [3], deploying the second sketch within INDDoS+ poses a new challenge. The three options are to deploy the second sketch in parallel to the first one, to use Tofino's folded pipeline, concatenating two processing pipes, or moving to newer switches (e.g., Tofino 2) with more processing stages. Therefore, hardware resources are not expected to be a bottleneck to offload INDDoS+ on programmable switches, and we consider this a future work.

VII. Related work

A. Sketch-based DDoS detection on programmable switches

Numerous methods have recently been proposed to identify DDoS attacks in programmable switches using diverse metrics, which can be retrieved from sketches. For instance, detecting DDoS attacks through the reduction in normalized entropy across distinct destination IP addresses (e.g., [14] [15]) uses sketches to estimate the packet count of each flow, such as Count-min Sketch [16] and Count Sketch [4]. Alternatively, it can involve detecting a substantial volume of unique data flows (using same source IPs) directed towards a particular destination host (e.g., [17] [2]), commonly referred to as per-destination *flow cardinality* (called connections in this paper). Many sketch-based algorithms for estimating the cardinality of data streams have been proposed, including Multiresolution Bitmap [5], PCSA [9] and HyperLogLog [10]. Nonetheless, without protection, values in sketches may be tampered by attackers, leading to large bias on the estimations in both servers and collectors. Inaccurate estimations significantly degrade DDoS detection performance.

B. Adversarial model of Sketches

While sketches have been widely used for network monitoring, protecting them is a new challenge. Reviriego *et al.* [18] first investigated the security issues of Count-min Sketch, generating fake elephant flows, that is, the flows with pretend to have a large number of packets. Recently, many works [12] [19] [11] started studying the potential vulnerabilities of HyperLogLog Sketch, where attackers can exploit such vulnerabilities to mislead the results of HyperLogLog. Unlike other related work, this work focuses on the security of sketch (i.e. BACON Sketch) deployed in high-performance programmable switches to detect the network anomalies. We comprehensively analyzed the potential issues of BACON Sketch and proposed corresponding strategies to defend against them. Importantly, the proposed inflating and evasion attacks can be generalized to other sketches, leading to potential network vulnerabilities.

VIII. Conclusion

In this paper, we studied two possible vulnerabilities of BACON Sketch and constructed attacks based on them. To detect and mitigate such attacks, we proposed a novel approach called INDDoS+, improving the resilience of BACON Sketch while maintaining the reliability of DDoS detection mechanisms in high-performance programmable switches.

Acknowledgements

This research was partly funded by EU Horizon SmartEdge (101092908, Innovate UK 10056403), VMWare Research and Oxford UNIQ+. For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript (AAM) version arising from this submission.

References

[1] R. Harrison *et al.*, "Network-wide heavy hitter detection with commodity switches," in *ACM SOSR*, 2018.
[2] Z. Liu *et al.*, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *ACM SIGCOMM*, 2016.
[3] D. Ding *et al.*, "In-Network Volumetric DDoS Victim Identification Using Programmable Commodity Switches," *IEEE Transactions on Network and Service Management*, 2021.
[4] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *VLDB*, vol. 1, no. 2, pp. 1530–1541, 2008.
[5] C. Estan *et al.*, "Bitmap algorithms for counting active flows on high speed links," in *ACM IMC*, 2003.
[6] J. S. Sobolewski, "Cyclic redundancy check," in *Encyclopedia of Computer Science*, 2003, pp. 476–479.
[7] R. Rivest, "Rfc1321: The md5 message-digest algorithm," 1992.
[8] P. Bosshart *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.
[9] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *Journal of computer and system sciences*, vol. 31, no. 2, pp. 182–209, 1985.
[10] P. Flajolet *et al.*, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Discrete Mathematics and Theoretical Computer Science*, pp. 137-156, 2007.
[11] P. Reviriego and D. Ting, "Security of HyperLogLog (HLL) cardinality estimation: Vulnerabilities and protection," *IEEE Communications Letters*, vol. 24, no. 5, pp. 976–980, 2020.
[12] K. G. Paterson and M. Raynal, "Hyperloglog: Exponentially bad in adversarial settings," in *IEEE EuroS&P*. IEEE, 2022, pp. 154–170.
[13] CAIDA UCSD Anonymized Internet Traces Dataset -[passive-2018], "http://www.caida.org/data/passive/passive_dataset.xml," 2018.
[14] K. Giotis *et al.*, "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.
[15] R. Wang *et al.*, "An entropy-based distributed DDoS detection mechanism in software-defined networking," in *IEEE Trustcom/BigDataSE/ISPA*, 2015.
[16] G. Cormode, "Count-min sketch," in *Springer Encyclopedia of Database Systems*, pp. 511-516, 2009.
[17] M. Yu *et al.*, "Software defined traffic measurement with OpenSketch," in *USENIX NSDI*, 2013.
[18] J. Murua and P. Reviriego, "Faking elephant flows on the count min sketch," *IEEE Networking Letters*, vol. 2, no. 4, pp. 199–202, 2020.
[19] D. Ding, "Carbine: Exploring additional properties of hyperloglog for secure and robust flow cardinality estimation," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, 2024.