Lemon: Network-Wide DDoS Detection with Routing-Oblivious Per-Flow Measurement

Wenhao Wu[†‡], Zhenyu Li[†‡*], Xilai Liu[†‡],
Zhaohua Wang[§], Heng Pan[§], Guangxing Zhang[†], and Gaogang Xie[§‡]

[†]*Institute of Computing Technology, Chinese Academy of Sciences*
[‡]*University of Chinese Academy of Sciences*
[§]*Computer Network Information Center, Chinese Academy of Sciences*

Abstract

Network-wide DDoS (Distributed Denial-of-Service) detection enables early attack detection and mitigates victim losses. However, unpredictable routing of DDoS traffic will invalidate the network administrator's prior knowledge of the network topology, causing existing sketch-based measurement systems to suffer from packet over-counting and processing stage mis-allocating issues. To address this gap, we propose Lemon, a routing-oblivious, resource-friendly, and scalable DDoS detection system that provides accurate detection of DDoS attacks without any assumption on the traffic routing. Specifically, we design a novel data structure (Lemon sketch) that supports over-counting-free and mis-allocating-free measurements in the data plane. Lemon control plane aggregates Lemon sketches from measurement points and leverages per-flow level network-wide measurement results for DDoS attack detection and victim identification. We implement Lemon in both software switch (Bmv2) and programmable switch hardware (Tofino). The evaluation results show that Lemon can achieve consistently high accuracy for DDoS detection in various topology and traffic distribution configurations.

1 Introduction

Currently, DDoS detection systems are usually deployed on a downstream server or switch that near DDoS traffic convergence links [25, 43, 47], where all attack traffic can be observed. However, such near-victim detection fails to respond to attacks before they reach the detector and wastes resources on the routing paths to the victim [35,53,54]. Therefore, detecting and mitigating DDoS traffic in upstream of convergence links is an ideal strategy. In this context, administrators (*e.g.,* ISPs) prioritize the overall state of the network rather than focusing on individual measurement points. By aggregating measurement results from multiple measurement points, administrators can obtain a comprehensive network-wide view, facilitating network-wide DDoS detection and minimizing victim losses.

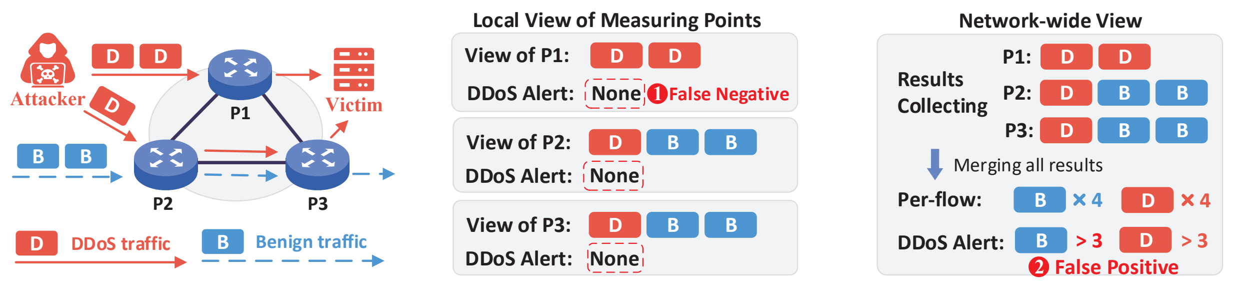
Sketch-based measurement is particularly promising for performing network-wide measurement. Rather than capturing full packets or flows, sketches (*e.g.,* CM-Sketch [17], Univmon [45], Elastic [60], Tower [58] *etc.*) record per-flow level information with compact probabilistic data structures. Since the low communication overhead and low resource consumption, Sketch-based DDoS attack detection is widely adopted by existing systems [19, 46, 52, 62] and performs well in resource-constrained scenarios (*e.g.,* deployed in programmable switches or end hosts [34,44]). To further improve the resource usage efficiency, some sketch designs [52, 60] adopt multi-stage processing, where traffic flows are allocated to different processing stages based on the flow size.

Existing sketch-based measurement systems [45, 46, 60] face significant challenges in practice when performing network-wide per-flow measurement. Specifically, measurement results are collected from multiple measurement points, necessitating the design of appropriate aggregation to prevent distortion of the measurement results. Existing solutions either assume the network's routing paths or require non-intersecting measurement results from multiple points. For instance, Univmon [45] assumes that the routing is prior knowledge and that each packet is only recorded by one point.

However, traffic routing is inherently dynamic, due to factors such as redundant multi-path [7], QoS-driven routing [12, 56], and traffic engineering [29, 31]. As a result, collecting exact routing information is prohibitively difficult in practice. The dynamic changing of routing invalidates the assumptions made by existing solutions and results in the following two issues that damage the accuracy of network-wide measurement:

- **Packet over-counting:** Packets traversing multiple measurement points are recorded multiple times, which leads to over-counting of packets when being aggregated for network-wide estimation.
- **Stage mis-allocating:** Individual measurement points only

[*]Corresponding author: Zhenyu Li.



(a) DDoS traffic from attackers enters the measurement system through multiple paths.

(b) DDoS traffic observed at each measurement point is less than the pre-defined threshold, no DDoS alert is triggered.

(c) Controller adds the measurement data from all measurement points, causing benign traffic exceed the DDoS threshold.

Figure 1: DDoS detection with network-wide measurement.

observe a small portion of traffic and thus may mistakenly allocate the globally heavy flow to the processing stage for small flows, leading to inaccuracy when aggregating.

To this end, obtaining an accurate network-wide view with unknown topology and traffic routing (*i.e.,* **routing-oblivious measurement**) is more promising. A viable solution is utilizing hash-based data structures, such as Hyperloglog [24] and coupon collector [15, 55], as in [9, 10, 20]. By doing so, a data packet will generate the same hash value at different measurement points, so only one valid update will be generated when passing through multiple measurement points (*e.g.,* updating the same bit of the bitmaps). Then the challenges nail down to design a hash-based sketch data structure that can accurately allocate processing stage according to global statistics (as opposed to local statistics in Couper [52]).

In this paper, we propose **Lemon**, a routing-oblivious, resource-friendly, and scalable network-wide DDoS detection system. To the best of our knowledge, Lemon is the first solution that provides the per-flow level routing-oblivious network-wide abstraction while maintaining resource friendliness and scalability. Central to our approach is the Lemon sketch, a novel data structure with multiple layers that takes the uniqueness of each packet to avoid packet over-counting and uses globally consistent update operations to avoid stage mis-allocating. Lemon sketch dynamically allocates units with different sizes to accommodate both small flows and heavy flows, enabling network-wide per-flow measurements with very limited on-chip memory consumption ($\approx$1MB). Lemon control plane aggregates measurement results to construct a precise network-wide view, thereby detecting network-wide DDoS attacks by leveraging Lemon's measurement outcomes. Extensive experiments with real-world traces demonstrate Lemon's effectiveness in DDoS attack detection.

In summary, our main contributions are as follows:

- We propose a network-wide DDoS detection system—Lemon, which can provide accurate detection of DDoS attacks without any assumption on network topology and traffic routing.
- We propose Lemon sketch for over-counting-free and mis-allocating-free per-flow measurement. Lemon sketch proves highly effective in enhancing network-wide DDoS attack detection.
- We implement the Lemon prototype and make it publicly available [1]. The experimental results show that, in comparison with the state-of-the-art solutions, Lemon improves the F1 score of up to 20% and achieves consistently high accuracy in various topology and traffic distribution.

2 Background and Motivation

2.1 Network-Wide DDoS Detection

Network-wide DDoS attack detection requires accurate measurement of all traffic that is distributed across multiple measurement points. However, the limited view of individual measurement points and the lack of aggregation schemes hinder the network-wide detection of DDoS attacks. We illustrate the challenges of network-wide DDoS attack detection in Fig. 1, where there are three measurement points in the network and the threshold for DDoS attack detection is 3[1]. Network-wide DDoS detection with measurement results from these measurement points suffers from two problems: ① False negative problem caused by limited local view: the attack traffic volume in the local view of each measurement point may be lower than the detection threshold and thus no DDoS alerts will be triggered. ② False positive problem caused by the lack of accurate aggregation: benign traffic volume may exceed the threshold due to over-counting the packets that traverse multiple measurement points.

To prevent errors in network-wide measurements, existing approaches typically rely on restrictive assumptions about network topology and routing. For instance, Univmon [45] (which is leveraged by Jaqen [46] as its data plane) requires

[1]For illustration purpose, we assume volume-based detection [11].

	Over-counting free	Mis-allocating free
Count-Less [36]	✗	✗
Jaqen [46]	✗	✓
Couper [52]	✓	✗
Lemon	✓	✓

Table 1: Comparison of network-wide measurement ability with existing sketch-based approaches.

packets to pass through only one measurement point, or, requires precise information of traffic routing to aid in the aggregation of measurement data by solving Integer Linear Programs (ILPs). However, accurately obtaining routing information is not an easy task since routing is highly dynamic in practice. Flows with the same source and destination may follow different paths because of load balancing or fault tolerance [7]. Additionally, flows may also take non-default paths to meet QoS requirements by user-side control [12, 56] or by traffic engineering from operators [29, 31]. As a result, routing can change frequently—every few minutes or on demand. Attackers can also exploit these mechanisms to evade detection. Motivated by these facts, the routing-oblivious measurement holds greater promise in network-wide DDoS detection.

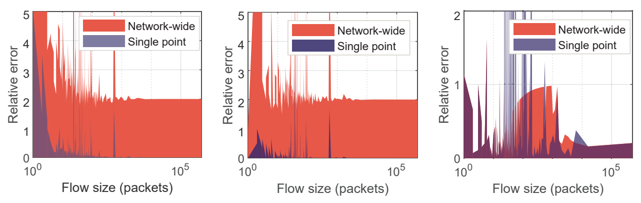
2.2 Motivation

We compare existing solutions for network-wide measurement in Table 1 and summarize the two major issues when applying them in practice.

Packet over-counting issue. In counter-based sketch systems [36, 46, 60, 62], each time a packet from a particular flow is processed, the corresponding counter is incremented. However, this approach neglects the uniqueness of each packet. When the same packet passes through multiple measurement points, it is counted independently at each point, leading to multiple counts of the same packet in the aggregated network-wide result. We show the per-flow measurement results of Jaqen (Fig. 2(a)) and CL-MU (Fig. 2(b)), both of which exhibit significant errors due to packet over-counting when performing network-wide measurement[2].

Stage mis-allocating issue. For optimized memory utilization, the state-of-the-art sketch designs (*e.g.,* Couper) [52] often allocate different processing stages based on the size of the flow. Specifically, small flows and heavy flows are processed in different stages with different counting mechanisms. For instance, in Couper [52] (see Fig. 3), small flows are initially tracked in the first layer using a bitmap, which requires only a minimal number of bits. Once the bitmap capacity is exceeded, the flow is transitioned to a HyperLogLog unit in the subsequent layer, which consumes more memory but has

[2]The measurement task estimates the volume of each destination IP with 5M packets from [16] in 100 measurement points. The network setup follows the setting in Accuracy vs. network scale of Section 6.4.



(a) Jaqen (Univmon). (b) Count-Less. (c) Couper.

Figure 2: Per-flow estimating error in network-wide measurement: Over-counting in Jaqen (a) and CL-MU (b); Mis-allocating in Couper (c).

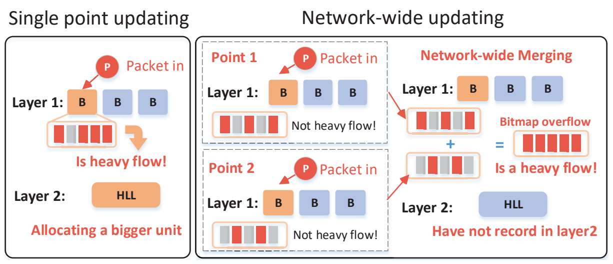


Figure 3: Network-wide measurement failed in Couper by mis-allocating issue.

a larger counting range. For a heavy flow that is sufficiently dispersed across the network, as in the case of DDoS attacks, multiple nodes may independently mis-allocate this flow to the stage that processes small flows with bitmap. Note that the decision to which stage that a flow is allocated is based on local statistics. As a result, during network-wide aggregation, the corresponding bitmap can become overwhelmed[3], as it potentially records too many elements of a distributed heavy flow. This leads to deviations between the measured results and the actual traffic volume, as shown in Fig. 2(c).

Motivated by the issues that lead to inaccurate network-wide DDoS detection, We design Lemon, a routing-oblivious, resource-friendly, and scalable DDoS detection system.

3 Lemon Overview

3.1 Problem Scope

Threat model : Our focus is on volumetric DDoS attacks, where attackers aim to exhaust the available bandwidth and resources of the victim by flooding with a large volume of DDoS traffic. In particular, we focus on scenarios in which attack traffic adopts multiple routing paths to the victim (and through multiple measurement points). The routing of attack traffic remains **unknown**—the measurement points traversed by each packet are unpredictable.

[3]A bitmap using linear counting algorithm will produce large estimating error when too many bits are set to ones

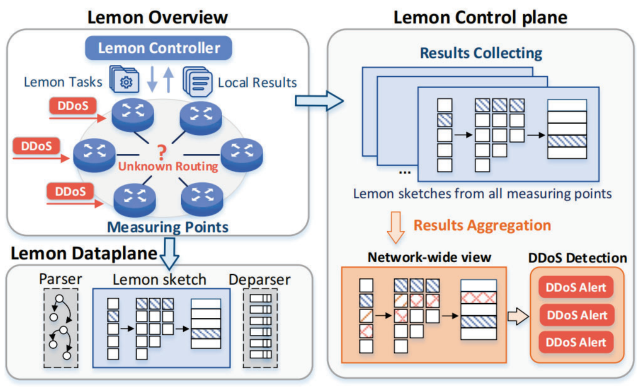


Figure 4: Overview of the Lemon system.

Deployment scenario : We consider deployment scenarios with multiple measurement points. We assume that the network measurement points are programmable and that there are sufficient measurement points to ensure each data packet is observed at least once.

3.2 Design Goals

Network-wide DDoS attack detection consists of a centralized control plane and multiple measurement points. Each measurement point conducts individual per-flow measurements and reports the results to the control plane. Utilizing data structures that enable routing-oblivious measurement in the data plane, the control plane aggregates these measurements and subsequently performs DDoS detection based on the aggregated data. It is worth noting that the control plane can provide specific configurations (*e.g.,* the IP ranges that are interested by the operator) to the data plane. We have the following design goals:

Goal 1: Routing-oblivious measurement. Network-wide DDoS detection needs accurate per-flow estimation. Appropriate data structure and corresponding aggregation algorithm are needed to overcome over-counting and mis-allocating issues, thereby enabling routing-oblivious measurement.

Goal 2: Resource-friendliness. Measurement points have limited hardware resources in the data plane and specific computing paradigm (*e.g.,* Intel-Tofino). We need to carefully design the data structure to effectively utilize the computing resource and meet the constraints of the programmable switch.

Goal 3: High scalability. The system should flexibly support typical measurement tasks and diverse detection algorithms [26] (*e.g.,* entropy-based and volume-based algorithms). Therefore, we need to design a control plane that can flexibly support the configurations of flow keys and the deployment of diverse DDoS attack detection algorithms.

3.3 Lemon Architecture

We design Lemon to meet all the design goals. As shown in Fig. 4, Lemon data plane consists of multiple measurement points, and Lemon control plane is a centralized controller for measurement results aggregation and DDoS detection.

Lemon data plane (Section 4) : The core of the data plane is the Lemon sketch. Lemon sketch uses hash-based volume estimation to consider the uniqueness of each data packet to enable over-counting-free measurement (meeting Goal 1). Under highly skewed network-wide traffic, Lemon can automatically allocate estimation units with different sizes for small flows and heavy flows without using any local statistics. By doing so, Lemon avoids the stage mis-allocating issue (meeting Goal 1) and thereby improves resource utilization efficiency (meeting Goal 2).

Lemon control plane (Section 5) : The centralized controller collects Lemon sketches from all measurement points, and aggregates them into a global Lemon sketch. The global Lemon sketch provides network-wide measurement results for various DDoS attack detection algorithms. Specifically, Lemon is compatible with configuration methods similar to existing sketch-based measurement systems and can provide reliable network-wide measurement results for flexible DDoS detection and attack volume estimation (meeting Goal 3).

4 Lemon Data Plane

Lemon's data plane records per-flow level information. Specifically, given a flow identifier[4], we can obtain the estimated volume of this flow from the Lemon sketch. Different from existing sketch-based measurement systems, Lemon sketch uses globally-consistent updating operations to support routing-oblivious measurement.

4.1 Preliminaries

Problem Definitions: Lemon data plane is composed of n network measurement points, where each point $i \in \{1,\ldots,n\}$ can observed a subset of packets $P_i = \{p_1, p_2, p_3, \ldots, p_n\}$ of the network-wide traffic P_{net}, where $P_{net} = \cup_{i=1}^{n} P_i$. Each packet p_i is represented as $\langle key_{flow}, key_{pkg}\rangle$, where key_{flow} is a flow identifier and key_{pkg} is a unique packet identifier[5] (*e.g.,* sequence number or checksum.). Our goal is to use P_i to estimate the packet count for each flow sharing the same key_{flow} in P_{net}.

Strawman solutions: Given that each packet is uniquely identified by key_{pkg}, an effective strategy to prevent over-counting involves reframing the packet counting as a per-flow

[4]The flow identifier can be flexibly defined according to the measurement task, *i.e.,* $\langle sIP, dIP, sPort, dPort, Protocol\rangle$ for per-flow volume estimation, $\langle sIP\rangle$ for source address frequency estimation.

[5]Previous works [21] [65] also suggest various methods to identify packets according to their header fields.

cardinality estimation problem. By employing a cardinality estimator for each flow, we can accurately get the packet volume (*i.e.,* the number of packets with different key_{pkg}) of key_{flow} in P_{net}. A strawman solution is to combine the Count-Min (CM) sketch with HyperLogLog to achieve per-flow cardinality estimating, where each unit of the CM sketch is replaced with a HyperLogLog structure. While this strawman solution prevents over-counting during aggregation by per-flow cardinality estimation, using HyperLogLog for every flow necessitates significant memory resources. For instance, employing a 262,144×2 size sketch with HyperLogLog structure of only 64 bytes leads to the memory usage of about 33MB, which is beyond the on-chip memory on typical Tofino switches.

Building on this insight, an advanced idea involves distinct counting mechanisms for heavy and small flows. Small flows are tracked within space-saving bitmaps. Once the bitmap's capacity is exceeded, the flow will recorded in a HyperLogLog unit subsequently. However, as we discussed in Section 2.2, such a solution (*e.g.,* Couper [52]) fails in providing accurate network-wide measurement tasks due to stage mis-allocating issue.

4.2 Lemon Sketch Design

The underlying problem that causes the failure of strawman solutions is its reliance on local statistics (*i.e.,* the flow size measured at a specific measurement point) to differentiate between heavy and small flows. Motivated by this observation, we adopt a globally consistent approach to distinguish the type of flows. The core idea of Lemon sketch is hash-based sampling, where the allocation of sketch units fully depends on the hash result of each packet. Lemon sketch leverages only hash results to adaptively allocate small flows and heavy flows in different units (with different sampling rates).

Lemon sketch consists of multiple layers from $layer_1$ to $layer_{n_{layer}}$, as shown in Fig. 5. Each $layer_i$ consists of S_i^s units, where each unit is a coupon collector (bitmap for linear counting [55]) with S_i^b bits. Lower layers (with smaller i) have a greater number of units compared to higher layers, with a higher probability of packet entry into lower layers. Consequently, small flows will be estimated in lower layers, while heavy flows will be estimated in higher layers. Additionally, the potential heavy-hitter key_{flow} will be recorded in a hash table $Heavy$. Lemon Sketch implements a globally consistent update (detailed in Section 4.3) and adaptive estimator reconstruction (detailed in Section 4.4) to restore different-sized estimators for different-sized flows, while mitigating the error caused by unit sharing (detailed in Section 4.5).

In Lemon, all measurement points use the same hash functions for Lemon sketch updating. As such, the updates caused by the same packet in different measuring points are the same. Such property enables over-counting-free measurement and mis-allocating-free aggregation in the control plane.

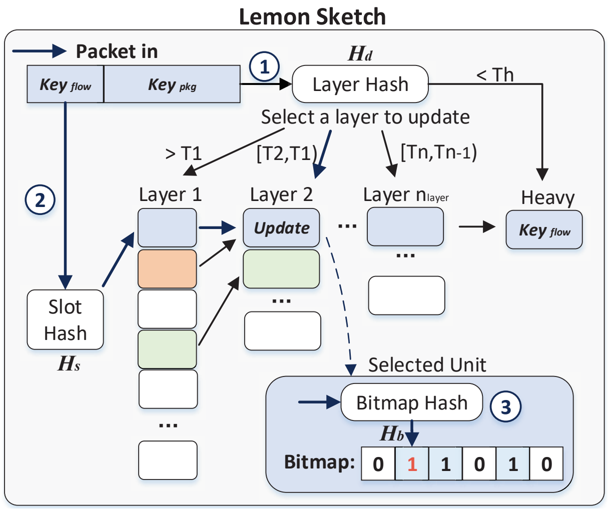


Figure 5: Lemon sketch design and updating.

4.3 Globally Consistent Updating

As shown in Fig. 5, for each packet with $\langle key_{flow}, key_{pkg}\rangle$, Lemon sketch uses three hash functions to determine how this packet updates Lemon sketch, where H_d is used to select the layer to be updated, H_s selects the unit to be updated in the layer, and H_b is used to update the bitmap bit in the unit. ① Lemon first uses H_d to hash key_{pkg} and obtain a hash value h_{layer} in the range $[0, H_d.max()]$, where $H_d.max()$ is the maximum value of H_d. Each $Layer_i$ in Lemon maintains a corresponding hash threshold T_i (where $T_0 = H_d.max()$), and each T_i of $layer_i$ always satisfies:

$$T_{i-1} > T_i \tag{1}$$
$$T_{i-2} - T_{i-1} > T_{i-1} - T_i \tag{2}$$

Such constraint signifies that higher layers have a smaller sampling rate. When $h_{layer} \in [T_i, T_i - 1)$, Lemon sketch will update $layer_i$. ② Lemon uses H_s to hash key_{flow} to obtain a hash value h_{slot} to determine which unit in the layer to update. All layers use the same hash function, and finally $h_{slot} \ \% \ S_i^s$ is used to get the unit that needs to be updated. Consequently, multiple units in lower layers sharing the same $h_{slot} \ \% \ S_i^s$ will collectively update the same unit in a higher layer (We mitigate such collisions caused by unit sharing, detailed in Section 4.5). ③ In the selected unit, Lemon uses H_b to hash key_{pkg} to update the corresponding bit in the bitmap.

Algorithm 1 shows the update process of Lemon sketch. Lemon first performs hash calculation (line 1-3), then finds the layer to be updated according to h_{layer} (line 4-5), and selects the corresponding unit in the layer to update the bitmap (line 6-9). If the hash value of the data packet is less than T_h, Lemon stores the key_{flow} of this packet in $Heavy$ (line 11-14).

```
Algorithm 1: Lemon sketch updating
    Input: Packet with flow key $key_{flow}$ and packet key $key_{pkg}$
    Output: Updated Lemon sketch
 1  $h_{slot} \leftarrow H_s(key_{flow})$
 2  $h_{layer} \leftarrow H_d(key_{pkg})$
 3  $h_{bitmap} \leftarrow H_b(key_{pkg})$
 4  for each layer of Lemon sketch $layer_i$ do
 5      if $h_{layer} \in [T_i, T_{i-1})$ then
 6          $index_{slot} \leftarrow h_{slot}$ % $S_i^s$
 7          $index_{bitmap} \leftarrow h_{bitmap}$ % $S_i^b$
 8          $bitmap \leftarrow layer_i[index_{slot}]$
 9          if $bitmap[index_{bitmap}] == 0$ then
10              $bitmap[index_{bitmap}] \leftarrow 1$
11          end
12      end
13  end
14  if $h_{layer} \leq T_h$ then
15      $index_{slot}^h \leftarrow h_{slot}$ % $Heavy.size$
16      $Heavy[index_{slot}^h] \leftarrow key_{flow}$
17  end
```

4.4 Adaptive Estimator Reconstruction

Given a flow identifier key_{flow}, Lemon queries the Lemon sketch to get the estimated volume E of the flow with key_{flow}. As shown in Fig. 6, ①Lemon first retrieves the corresponding unit of key_{flow} in all layers, resulting in n_{layer} bitmaps, each of which in $layer_i$ has a sampling rate of $\frac{T_{i-1}-T_i}{H_d.max()}$. ②Next, Lemon checks the number of zeros for bitmaps from $layer_i$ bottom-up, donated as Z_i and finds out the first bitmap that satisfies $Z_i \geq T_z \cdot S_i^b$, where T_z is a predefined threshold[6]. The final estimated result E is expressed as:

$$E = -S_i^b \cdot ln(Z_i/S_i^b) \cdot \frac{H_d.max()}{(T_{i-1} - T_i)} \tag{3}$$

We call the $layer_i$ of this bitmap the estimated layer, and $\frac{T_{i-1}-T_i}{H_d.max()}$ is called the *best sampling rate* of this flow. If no bitmap satisfying the condition is found, it indicates that the estimated value of the current flow exceeds Lemon's maximum estimation range. This situation can be avoided by setting a smaller sampling rate or larger bitmap size.

Algorithm 2 shows the query process. Lemon sketch first gets units from each layer for key_{flow} (line 1-9). Then it starts from the lowest layer and searches for the first bitmap that meets the condition (line 10-11). When it finds such a bitmap, it calculates the final estimated value E (line 12-17). If no such bitmap is found, it means that all layers have recorded too many elements, and overflow is returned (line 19).

[6]We set T_z to 0.25 to avoid saturated bitmaps [52].

```
Algorithm 2: Lemon sketch query for flow size
    Input: Flow identifier $key_{flow}$
    Output: Estimated flow size $E$ and allocated layer $L$
 1  $h_{slot} \leftarrow H_s(key_{flow})$
 2  $bitmaps \leftarrow i$ size empty list
 3  $T_{bitmap} \leftarrow$ pre-defined threshold
 4  for each layer of Lemon sketch $layer_i$ do
 5      if $h_{layer} \in [T_i, T_{i-1})$ then
 6          $index_{slot} \leftarrow h_{slot}$ % $S_i^s$
 7          $bitmaps[i] \leftarrow layer_i[index_{slot}]$
 8      end
 9  end
10  for $bitmaps[i]$ in bitmaps: $i = 1 : n_{layer}$ do
11      $Z_i \leftarrow bitmaps[i].Count\_0()$
12      if $Z_i \geq T_z \cdot S_i^b$ then
13          $E \leftarrow -S_i^b \cdot ln(Z_i/S_i^b)$
14          $E \leftarrow E \cdot \frac{(T_{i-1}-T_i)}{H_d.max()}$
15          $L \leftarrow i$
16          return $E, L$
17      end
18  end
19  return $overflow$
20  # avoiding by setting large enough bitmap for $layer_n$
```

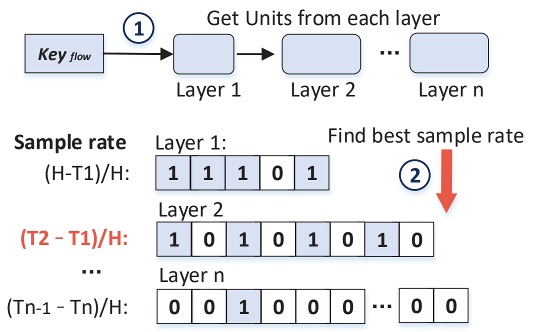


Figure 6: Lemon sketch querying.

4.5 Collision Elimination

In Lemon sketch updating, multiple units in lower layers sharing the same h_{slot} % S_i^s will collectively update the same unit in a higher layer. This results in collisions between multiple flows. We demonstrate the method to eliminate errors arising from these collisions. As shown in Fig. 7, multiple different units in $layer_{i-1}$ share the same unit in $layer_i$. This means flows f_1 and f_2 in ① will obtain the same unit in $layer_i$ when being queried. This collision can be eliminated when the best sampling rates of f_1 and f_2 are different (that is, f_1 and f_2 have different flow sizes and different estimated layers). Without loss of generality, we assume that ②f_1 is a larger flow with the best sample rate at $layer_i$, while ③f_2 gets the best sample rate at $layer_{i-1}$. In the estimation result, the estimated value of f_2 at a lower layer will not be affected by f_1, and

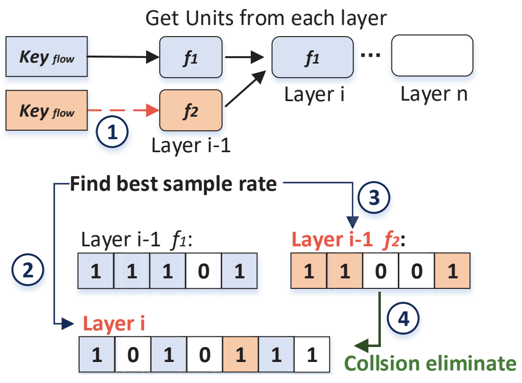


Figure 7: Eliminate collisions due to unit sharing.

the estimated value of f_1 is the sum of f_1 and f_2. We can eliminate this error by subtracting f_2 from the estimated value of f_1[7]. Note that collisions in f_1 cannot be eliminated if f_1 and f_2 are estimated in the same layer. However, as we will show in Section 4.6, the occurrence of this case is very rare.

Algorithm 3 shows how to eliminate collisions caused by unit sharing. For a flow estimated in $layer_i$, we identify the index of all potentially shared units in $Layer_{i-1}$ (line 1-7). Then eliminate the error from the estimated value of each unit in $Layer_{i-1}$ (line 8-13).

Algorithm 3: Eliminate collision

```
Input: Flow identifier $key_{flow}$
Output: Collision eliminated Estimated result $E$
1  $E, L \leftarrow Query(key_{flow})$
2  $h_{slot} \leftarrow H_s(key_{flow})$
3  for $i \in [0, \frac{S^s_{L-1}}{S^s_L})$ do
4      $index_{slot} \leftarrow i \cdot S^s_L + h_{slot} \ \% \ S^s_L$
5      if $index_{slot} == h_{slot} \ \% \ S^s_{L-1}$ then
6          continue
7      end
8      $bitmap \leftarrow layer_{L-1}[index_{slot}]$
9      $Z \leftarrow bitmap.Count\_0()$
10     if $Z \geq T \cdot S^b_{L-1}$ then
11         $E_{collision} \leftarrow -S^b_{L-1} \cdot log(Zeros/S^b_{L-1})$
12         $E \leftarrow E$ - $E_{collision}$
13     end
14 end
15 return $E$
```

4.6 Analysis

Since the flow collision rate directly affects the accuracy of the Lemon sketch, we analyze the collision rate of each layer. We care about in which layer will a flow be estimated and the probability of collision within each layer.

In which layer will a flow be estimated? For this question, we have the following theorem.

Theorem 1. *For a flow key_{flow} with N packets, its has a probability of Pr_i that get the best sample rate in $layer_i$ of the Lemon sketch, where Pr_i can be calculated as:*

$$Pr_i = (1 - \sum_{j=0}^{i-1} Pr_j) \cdot (1 - \sum_{j=0}^{\lfloor T_z \cdot S^b_i \rfloor} \binom{j}{S^b_i} B^j \cdot (1-B)^{S^b_i - j}) \quad (4)$$

where

$$B = (1 - \frac{1}{S^b_i})^{N \cdot \frac{(T_{i-1} - T_i)}{H_d.max()}} \quad (5)$$

Proof. The flow's estimated layer is $layer_i$ only if two conditions are met: 1) the bitmap in the corresponding unit in $layer_i$ has enough zeros that $Z_i \geq T_z \cdot S^b_i$ after the update is completed. 2) the flow does not get best sample rate in lower layers.

For the first condition, Since each packet is updated independently. In the corresponding bitmap of $layer_i$, the probability of the x-th bit is 0 after N_i times updated is:

$$B = Pr[bitmap[x] = 0] = (1 - \frac{1}{S^b_i})^{N_i} \quad (6)$$

where S^b_i is the size of bitmap in $layer_i$. N_i is the number of packets sampled into $layer_i$. According to the sampling rate of the i-th layer, N_i can be expressed as $N \cdot \frac{(T_{i-1} - T_i)}{H_d.max()}$.

The number of zeros in bitmap follows a binomial distribution. Therefore, the probability that the number of zeros Z_i in the bitmap is greater than $T_z \cdot S^b_i$ when update is finished is:

$$Pr[Z_i > T_z \cdot S^b_i] = (1 - \sum_{j=0}^{\lfloor T_z \cdot S^b_i \rfloor} \binom{j}{S^b_i} B^j \cdot (1-B)^{S^b_i - j}) \quad (7)$$

For the second condition, the probability that the flow has not been estimated in a lower layer can be expressed as:

$$Pr[not\ in\ lower\ layer] = (1 - \sum_{j=0}^{i-1} Pr_j) \quad (8)$$

where Pr_j is the probability that the flow gets best sample rate in $layer_j$. When the above conditions are met, the flow will be estimated at $layer_i$. Since each packet is updated independently, Pr_i can be calculated as the product of Eq. (7) and Eq. (8).

□

What is the probability of collision in each layer? For this problem, the number of flows mapped to each unit of a layer

[7] In practice, the estimated value of f_2 also includes the collision flows at $layer_{i-2}$ (record as f_{others}). We require that S^s_{i-1} is set as the integer multiples of S^s_i to ensure f_{others} also shares the same unit with f_1 at $layer_i$

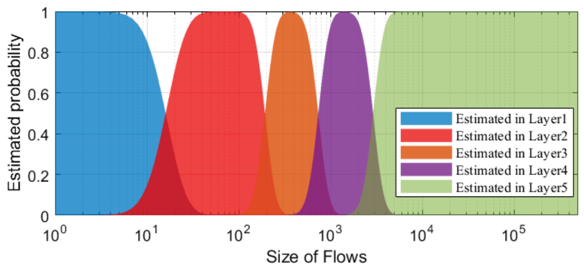


Figure 8: Probability of flow estimated in each layer.

follows a binomial distribution. Since this is a classic problem in the sketch algorithm, we omit the proof process here. We refer to the analysis in [60], and get the collision rate as:

$$Pr[collision\ in\ layer_i] = 1 - (\frac{N_i}{S_i^s} + 1)e^{-\frac{N_i}{S_i^s}} \tag{9}$$

where N_i represents the number of flows with estimated layers greater than $Layer_i$ (include $layer_i$) and S_i^s represents the number of units. For flows estimated in lower layers, their influence on $Layer_i$ can be eliminated. Under proper parameter configuration, the collision rate can be kept at a low level (less than 0.5% when $\frac{N_i}{S_i^s}$ is 0.1).

Case study: To illustrate the analysis result more clearly, we analyze the Lemon sketch with the following configuration: Lemon sketch with 5 layers, where T_i holds in each layer is 16,384, 4,096, 1,024, 256, and 0; The number of units S_i^s in each layer is 524,288, 65,536, 8,192, 2,048, and 1,024; The bitmap size S_i^b of each layer is 8, 32, 32, 32, and 512. Which is the same configuration as we used in the evaluation (in Section 6). We show how flows of different sizes are separated in this Lemon sketch. As shown in Fig. 8, Lemon sketch effectively segregates flows with different sizes into different layers (*e.g.,* flows with size over 10k are estimated in $layer_5$ and with size less than 10 is estimated in $layer_1$).

We use such Lemon sketch to conduct volume estimation for each destination IP with 5M packets from MAWI [16]. There are over 200k different destination addresses that need to be estimated and a few with large volumes (over 600k packets). In the aforementioned Lemon sketch, the number of flows in each layer (estimated in this layer or greater than this layer) is 222,470, 6,463, 360, 183, and 99; and the probability of collision at each layer is 7.2%, 0.55%, 0.29%, 0.86%, and 0.44%. It is demonstrably clear that the collision rate at each layer is minimal, with the incidence of collisions beyond the initial layer being negligible (collision rate is below 1%).

5 Lemon Control Plane

Lemon control plane consists of a centralized controller. The controller collects Lemon sketches from all measurement points in the network-wide and aggregates them into a global Lemon sketch. We then explain how the Lemon controller configures measurement tasks.

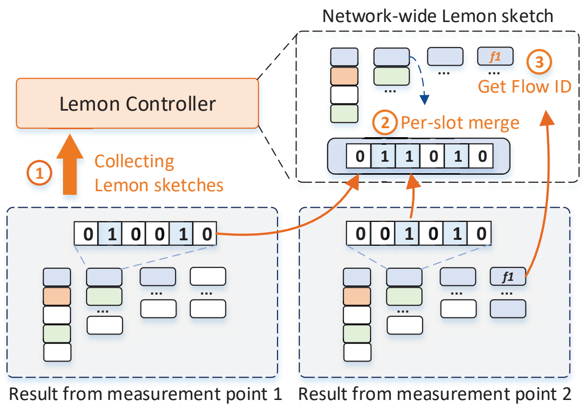

Network-wide Lemon sketch
Lemon Controller
① Collecting Lemon sketches
② Per-slot merge
③ Get Flow ID
f1
0 1 1 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
Result from measurement point 1
Result from measurement point 2

Figure 9: Aggregation for two measuring points.

5.1 Results Collecting and Aggregation

Similar to other sketch-based DDoS detection systems [59], the Lemon controller collects data from various measurement points at regular time intervals. We refer to such time interval as a measurement epoch. At the end of each measurement epoch, Lemon controller collects local Lemon sketches from measurement points and merges the sketches into a global Lemon sketch. It then detects potential DDoS attacks from the past epoch and identifies the victim IP addresses.

Fig.9 illustrates the process of aggregating Lemon sketches from measurement points to a global Lemon sketch. The controller performs the following steps: ①Collects all Lemon sketches from the data plane. ②For each layer in the Lemon sketch, the controller applies an OR operation to bitmaps with the same layer and same unit index, then places the result into the corresponding position of the global Lemon sketch. ③For the key_{flow} kept in *Heavy*, we directly save key_{flow} to global Lemon sketch. If the flow key stored in Lemon sketches of different measurement points differs[8], we estimate the local volume for each flow key and incorporate the flow key with the larger estimated value into the network-wide view. Since the same hash functions are used across all measurement points, the same data packets have the same updating behavior in each measurement point, thereby avoiding the over-counting problem in Lemon sketch aggregation.

5.2 Measurement Tasks Configuration

We illustrate the Lemon control flow and configuration method in Fig.10. Lemon deploys DDoS detection tasks by modifying the definition of key_{flow} and key_{pkg} in the data

[8]Different flow key means flow collision in the *Heavy*, and the collision rate is low because most of the flows do not sample to *Heavy*.

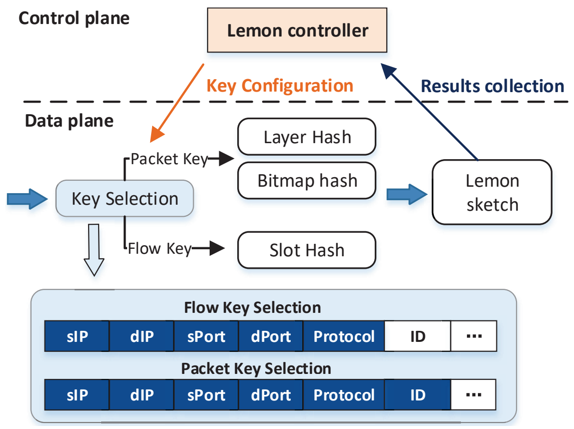


Figure 10: Measurement tasks configuration.

plane. Specifically, Lemon uses a set of registers in the data plane to maintain a mask for each field in the packet header. The control plane only needs to modify the value of the mask in the register to implement customized key_{flow} and key_{pkg}. For example, when measuring the volume of each flow, Lemon controller sets the five-tuple (source address, destination address, source port, destination port, protocol) to be valid in the mask of the flow key and set the packet key as a unique packet identifier[9].

Lemon can support typical network measurement tasks by configuring specific key_{flow} and key_{pkg}, including per-flow volume estimation, per-flow cardinality estimation, entropy estimation, etc. In the following section, we detail the configurations for these measurement tasks and demonstrate measurement-based DDoS attack detection.

5.3 DDoS Detection

DDoS attacks alter statistical indicators in the network [49], and detecting changes in these indicators enables DDoS detection. Lemon offers flexible support for statistics-based DDoS detection methods. We present three typical use cases of Lemon, along with the intuitions behind them.

Volume-based detection. A basic characteristic is that the victim experiences an excessive influx of DDoS packets from the network. Therefore, the volume-based detection method identifies DDoS victims subjected to abnormally high packet volumes, which is widely adopted in sketch-based measurement systems [46] [23] [4] [51]. Lemon configures the key_{flow} as the destination IP and key_{pkg} as the per-packet unique identifier, and stores the potential victim with high volume in *Heavy* (with the sample rate defined by T_h).

Cardinality-based detection. For DDoS attackers that exploit distributed hosts (*e.g.,* botnet [6]) to send traffic to the victim. The cardinality of unique source IP contacting with the same destination IP will increase, and DDoS victims contacted by an unusually high number of source IPs are identified [19]. Lemon configures the key_{flow} as the destination IP and key_{pkg} as the source IP, and stores the potential victim in *Heavy* (with sample rate defined by T_h). This configuration is similar to volume-based detection in Lemon. However, we distinguish them because previous sketch-based systems require entirely different data structures for these two tasks (*e.g.,* CM-Sketch for volume estimation and HyperLogLog for cardinality estimation).

Entropy-based detection. DDoS traffic from attackers will increase the entropy of the source address [18]. We can identify the change in the entropy to detect DDoS attacks effectively [18]. To achieve this, key_{flow} is configured as the source IP. Lemon traverses all possible h_{slot}, and then performs subsequent query operations (line 4-18 in Algorithm 2). Afterward, the address frequency distribution can be obtained by MRAC algorithm [38]. Finally, The entropy of the source IP can be calculated from the address distribution.

In addition to the above detection methods, Lemon's measurement results can flexibly support existing mechanisms (*e.g.,* combination of multiple measurement indicators), which we discuss in Section 7.

5.4 DDoS Mitigation

In-network mitigation. Lemon directly perform rate-limiting and drop from native programmable switch operation [27] like existing network measurement systems [46]. Lemon's DDoS attack mitigation depends on the configuration of the control plane access control list. For example, with volume-based DDoS detection, Lemon directly performs rate limit for the traffic to the victim (destination IP as flow keys) or mitigates DDoS traffic by dropping flows from high-volume sources (source IP or source-destination IP pairs as flow keys).

Integration with existing mechanisms. Lemon can be leveraged to enhance existing victim-near DDoS mitigation and traffic cleaning mechanisms. In practice, network managers usually deploy traffic-cleaning services in specific points [41, 50]. Lemon acts as an upstream measurement system, and only reroutes suspicious traffic to the downstream infrastructure that performs traffic cleaning (*e.g.,* with deep learning-based packet classification [5, 33, 61]), which significantly reduce the amount of traffic that needs to be cleaned. Some victim neared mechanisms (*e.g.,* Poseidon [62], Bohatei [22]) also assume that the DDoS victim is known a priori. Such mechanisms benefited from Lemon's early detection results to get the network status and potential victims.

[9]We use the checksum of the IP header and TCP/UDP header, along with the first 16 bits of the payload, as a per-packet identifier.

6 Evaluation

We have built a Lemon prototype with P4 in Bmv2 (about 470 lines of P4) and in programmable switch hardware [2] (about 580 lines of P4), and implemented Lemon control plane with Python. We evaluated the performance of Lemon using a variety of DDoS attack scenarios. We summarize the following experimental results:

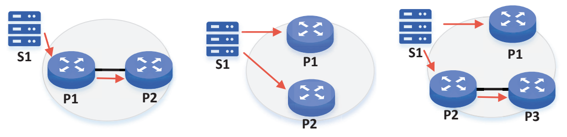
- Lemon is able to achieve effective DDoS attack detection (F1> 0.9 with 10% DDoS attack volume) and accurate attack volume estimation (relative error< 0.02) in network-wide DDoS detection, demonstrating robust performance over different network topologies (Section 6.2).
- DDoS detection performance of Lemon remains robust against address spoofing. The incorporation of multiple detection methods (*i.e.,* cardinality-based and entropy-based approaches) can enhance the effectiveness in identifying DDoS attacks (Section 6.3).
- Lemon achieves better performance in per-flow volume estimation with different flow distributions and flow keys and shows consistent performance under different network scales (Section 6.4).
- Lemon can meet the resource constraints of programmable switch ASIC with minimal resource consumption (0.35% TCAM and 10.31% SRAM) (Section 6.5).

6.1 Experimental Setup

Background traffic. We generate mixed traffic of background traffic and DDoS attacks to evaluate the DDoS detection performance of Lemon. We utilized traces from real-world ISP networks [16] as background traffic, encompassing over 5 million packets per epoch (about 7Gbps background traffic for the measurement epoch lasting 5 seconds). There are approximately 200k unique destination IP addresses in one measurement period.

DDoS attack traffic. Following the evaluation setup of prior studies [46], we conducted UDP flooding attacks and varied the volume of attack traffic (ranging from 0.1% to 10% of total traffic). For the experiments in Section 6.2, we launch attacks against the same victim address (randomly selected in the background traffic), and the attack traffic is generated with random sources (each packet source is randomly chosen from public IP addresses). For the genericity experiments in Section 6.3, we modified the number of attack sources to investigate the impact of source spoofing (or botnet) on victim detection and modified the victim (from a single victim IP to subnets) to evaluate the impact of destination spoofing (*i.e.,* carpet bombing [13, 32]).

Topologies and measurement points. Mixed traffic will pass through multiple measurement points of Lemon (with routes unknown to the control plane). All nodes in the network perform as the measurement points since measurement systems



(a) **[T2]** Serial. (b) **[T3]** Parallel. (c) **[T4]** Mixed.

Figure 11: Controlled topologies **[T2]-[T4]**. S1 distributes mixed traffic to measurement points P1-P3.

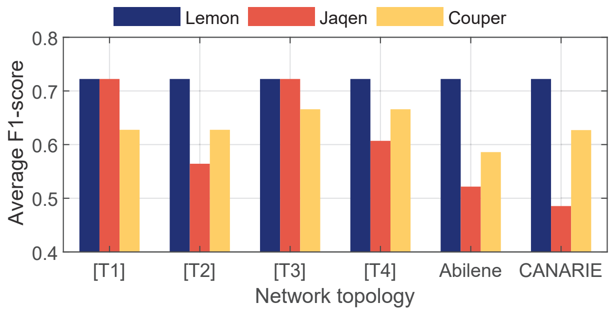


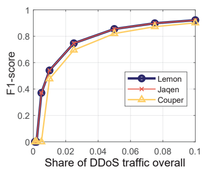
Figure 12: Comparison of DDoS detection performance.

are typically integrated with routing devices[10]. We designed the following topology configure and traffic routing scenarios, including controlled network topologies (**[T1]-[T4]**) to illustrate the cause of error in DDoS detection; and the scenarios with real-world ISP topologies (Abilne and CANARIE).

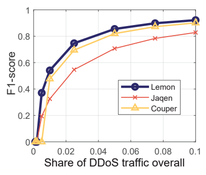
- **[T1]**: Single point. There is only one measurement point in the network and all packets will pass through this point once.
- **[T2]**: Serial points. There are two serial points organized as in Fig. 11(a), and each packet will be observed by both measurement points, resulting in over-counting issue.
- **[T3]**: Parallel points. There are parallel measured points as in Fig. 11(b). Packets will enter the two measurement points evenly, each point only observes a part of the traffic.
- **[T4]**: Mixed situation. The topology consists of three points as shown in Fig. 11(c). This configuration is the simplest combination of parallel points and serial points.
- **Abilene [37]**: Abilene topology has 12 measurement points. Each packet randomly selects a point to enter the network and be randomly forwarded (with hop limit is 5).
- **CANARIE [14]**: CANARIE topology has 28 measurement points. Each packets randomly select a point to enter the network and be randomly forwarded (with hop limit is 15).

Evaluation metrics: DDoS detection performance is measured with *F1-score (F1)* [28]. We also use *Relative Error (RE)* to evaluate the accuracy of attack volume estimation.

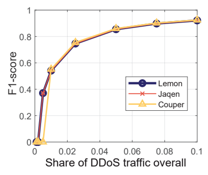
[10]For scenarios that only part of nodes are measurement points, each packet (to be measured) should pass at least one measurement point.



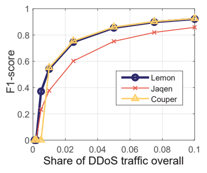
(a) F1 in **[T1]**.



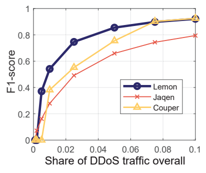
(b) F1 in **[T2]**.



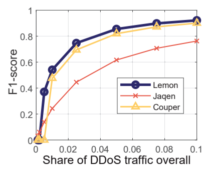
(c) F1 in **[T3]**.



(d) F1 in **[T4]**.

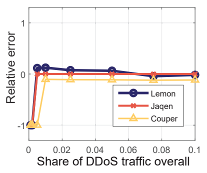


(e) F1 in Abilene.

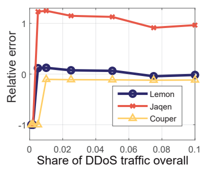


(f) F1 in CANARIE.

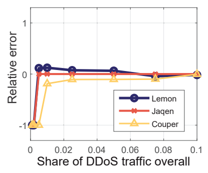
Figure 13: DDoS detection performance.



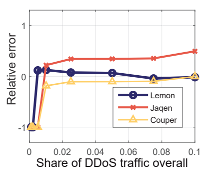
(a) Error in **[T1]**.



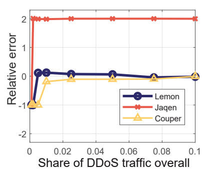
(b) Error in **[T2]**.



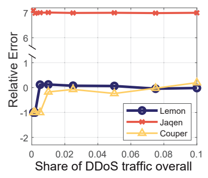
(c) Error in **[T3]**.



(d) Error in **[T4]**.



(e) Error in Abilene.



(f) Error in CANARIE.

Figure 14: Attack volume estimating error.

We use Weighted Absolute Percentage Error (WAPE) [11] to evaluate the accuracy of per-flow volume estimation.

Baselines : We compare Lemon with the state-of-the-art sketch based solutions, including:

- Jaqen [46]: an UnivMon-based DDoS detection system. It uses volume-based DDoS detection (*i.e.,* to check whether the number of packets with the specified flag exceeds the pre-defined threshold). We get a network-wide view by summing Jaqen measurement results.
- Couper [52]: a data structure that supports per-flow cardinality estimation. It uses bitmaps to estimate small flows, and HyperLoglog to estimate heavy flows. The bitmap is set as 16 bits, and the size of each HyperLoglog is 64 bytes. We set the number of bitmaps to be consistent with $layer_1$ of Lemon to ensure enough bitmaps to handle small flows and set the number of HyperLoglog consistent with $layer_5$ of Lemon for estimating heavy flows.

Settings and DDoS detection method : In our experiments,

[11]Ratio of Mean Absolute Error (MAE) and Mean value of ground truths.

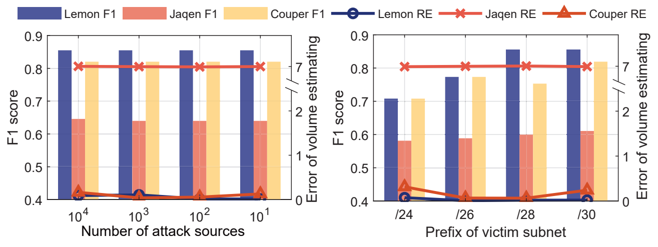


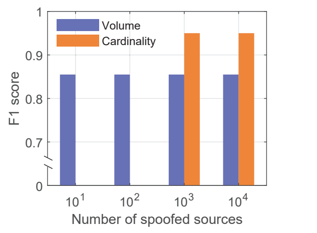
Figure 15: DDoS detection performance under source spoofing and carpet bombing.

Lemon sketch is set with five layers. Where T_i for each layer is 16,384, 4,096, 1,024, 256, and 0; The threshold T_h is 256 to record the $flow_{keys}$ that estimated in $layer_5$; The number of units for each layer S_i^s is 524,288, 65,536, 8,192, 2,048, and 1,024; The size of bitmap in each layer S_i^b is 8, 32, 32, 32, and 512. We set key_{flow} as the destination IP and key_{pkg} as the unique per-packet identifier. Unless otherwise specified, Lemon and all baselines adopt volume-based DDoS detection, with the same detection threshold as 0.5% (Following settings in Jaqen). Note that the performance of DDoS detection is only affected by the accuracy of the network-wide measurement.

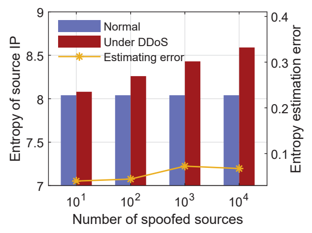
6.2 DDoS Detection Accuracy

Overall performance (Fig. 12). We first compare the overall detection accuracy in Fig. 12 for each scenario. The F1-score is the average value when the DDoS volume accounts for over 0.5%. Lemon consistently demonstrates the best detection performance, unaffected by changes in topology and routing configurations. Compared to Jaqen, Lemon performs similarly in scenarios **[T1]** and **[T3]**, as these scenarios do not involve over-counting. However, in scenarios **[T2]** and **[T4]** where over-counting is occurring, Lemon's attack detection performance is significantly better than Jaqen. The over-counting issue causes flows in background traffic to mistakenly exceed the DDoS detection threshold, resulting in higher false positives and a lower F1 score. The results of Jaqen deteriorate significantly as the network topology becomes more complicated in Abilene and CANARIE. Compared to Couper, Lemon demonstrates superior performance in all attack scenarios, which is attributed to the higher accuracy of per-flow volume estimating. Couper's error was primarily due to using local information to process flows (As we introduced in Section 2.2), which resulted in the stage mis-allocating issue.

Performance with attack volume (Fig.13). Generally, higher attack volume will make DDoS attacks easier to detect. Lemon consistently outperforms the other baselines in DDoS detection due to its accuracy in network-wide per-flow measurement. When the DDoS attack volume is 10%, Lemon achieves an F1 of over 0.9. Jaqen's error is caused by over-counting, which leads to excessive flow volume estimations,

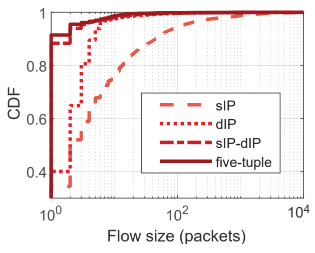


(a) Performance in volume-based and cardinality-based detection.

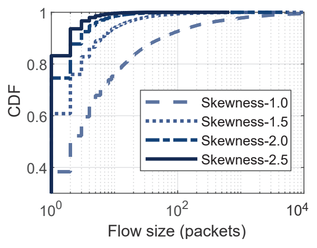


(b) Entropy of source IP under DDoS attacks.

Figure 16: Performance for different Lemon configurations.

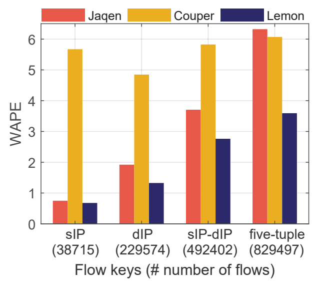


(a) CDF for different flow keys.

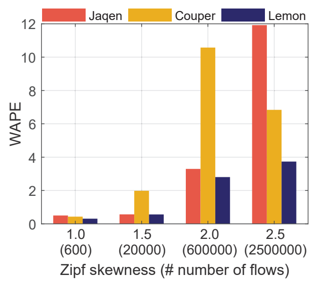


(b) CDF for different Zipf skewness.

Figure 17: Distribution of flow size.



(a) Performance in flow keys change.



(b) Performance in FSD change.

Figure 18: Per-flow measurement error.

with a remarkable performance drop in DDoS detection of **[T2]**, **[T4]**, Abilene, and CANARIE. Couper, on the other hand, suffers from stage mis-allocating issue, which is particularly pronounced in larger networks.

Performance of attack volume estimation (Fig.14). We further illustrate the relative error of attack volume estimation. Lemon consistently achieves accurate volume estimation of attack traffic within each scenario. In contrast, Jaqen exhibits significant over-counting issues in scenarios **[T2]**, **[T4]**, Abilene, and CANARIE, resulting in the estimated result of attack volume substantially higher than the actual values. While Couper performs more accurately than Jaqen and mitigates the over-counting problem to some extent, its estimates are smaller than the actual values because of mis-allocating.

6.3 Resilience to Address Spoofing

To evaluate the resilience of Lemon to IP-spoofed traffic, we consider the DDoS detection performance with source spoofing and destination spoofing (with CANARIE topology). The setups follow the pattern in the previous part and DDoS attack volume account for 5% of background traffic. We then modify the attack traffic by using: (i) Source spoofing. We modify the attack source, the number of spoofed IPs ranges from 10 to 10k; and (ii) Destination spoofing (*i.e.,* carpet bombing). The attack targets the /24 to /30 destination prefix instead of a single IP. The results are shown in Fig.15.

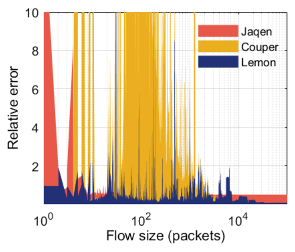
Source spoofing. The detection accuracy of all methods remained relatively stable since we used destination IP to find victims in DDoS detection. Lemon always achieves better performance against baselines. Nevertheless, source spoofing will affect certain detection methods. We further evaluate the impact of source spoofing on different detection methods. Fig.16(a) illustrates performance using the cardinality-based DDoS detection[12]. When the number of spoofed IPs is less than 100, cardinality-base approach cannot detect DDoS since the cardinality of attack sources does not significantly change. When the number of spoofed IPs exceeds 1,000, cardinality-based method outperforms the volume-based detection. Fig.16(b) demonstrates entropy-based detection, which reflects the overall network status without identifying specific victims. As the number of spoofed IPs increases, the entropy of source addresses rises significantly, which indicating DDoS can be more likely to be detected.

[12]Threshold is 0.05% of the total number of distinct sIPs, following [19].

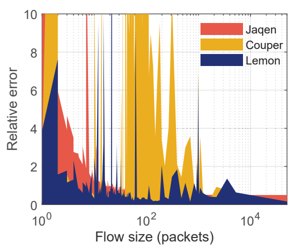
Destination spoofing. We counter destination spoofing by changing key_{flow} from destination IP to destination subnet (*e.g.,* for /24 carpet bombing, we estimate the volume of each /24 prefix). The performance of Lemon is not affected by the /28 and /30 carpet bombings. When the subnet is further expanded, the accuracy of all methods decreases. Compared with all baselines, Lemon can achieve better performance, which comes from accurate network-wide measurement.

6.4 Per-Flow Measurement Accuracy

As the number of flows in the network increases, the collision in Lemon sketch tends to rise. This section investigates the per-flow measurement performance of Lemon sketch under more complex flow keys and larger workloads. The experiments in this section are conducted in **[T4]** with: (i) Complex flow keys: including source address, destination address, source-destination IP pair, and five-tuple to perform per-flow volume estimation with 5M packets from MAWI [16]. (ii) Network traffic with different flow size distribution (FSD): We create traces that follow Zipfian (Zipf) flow size distribution with skewness from 1.0 to 2.5 (following the method in [36]), where each trace contains 5M packets. We use source



(a) Estimating error for sIP



(b) Estimating error for five-tuple

Figure 19: Estimating error with flow size.

IP addresses as flow key for volume estimation and the number of flows (distinct sources) reaches up to 2.5M (skewness of 2.5). The distribution of flow size with complex flow keys and generated traces are shown in Fig.17(a) and Fig.17(b), respectively.

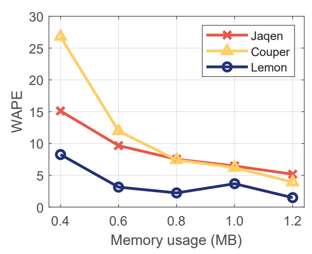
Accuracy vs. flow keys (Fig.18(a)). Lemon consistently demonstrates the lowest error across all flow key types. When the five-tuple is used as the flow key, the WAPE of all systems increases due to the increased collision rate. Nevertheless, Lemon achieves a much lower WAPE compared to Jaqen and Couper, demonstrating its scalability for different key sets.

Accuracy vs. traffic skewness (Fig.18(b)). Lemon consistently achieves the lowest error across all traffic skewness, highlighting its robustness in handling highly skewed traffic distributions. With the most challenging skewness of 2.5, where the number of flows is highest, Lemon maintains the lowest WAPE against baselines. This indicates Lemon's better performance under extreme conditions.

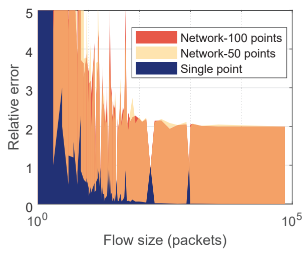
Accuracy vs. per-flow size (Fig.19). Fig.19 illustrates the per-flow estimating error under different flow key configurations (sIP and five-tuple). Lemon demonstrates a more significant advantage in the middle range of flow sizes against Couper. This is attributed to Lemon's mis-allocating-free property, which efficiently allocates resources to reduce conflict in mid-sized flows. Additionally, Lemon's multi-layer structure ensures that while small flows might experience conflicts in the lower layers, the accuracy for large flows in higher layers almost remains unaffected.

Accuracy vs. memory usage (Fig. 20(a)): We analyze the performance change with different memory usage by proportionally varying the number of units in each layer (with per five-tuple volume estimation of 5M packets from MAWI). Lemon consistently achieves better per-flow estimation accuracy. Notably, when the memory usage is small ($\leq$0.4MB), Lemon's WAPE reaches its maximum due to flow collision caused by the insufficient number of units.

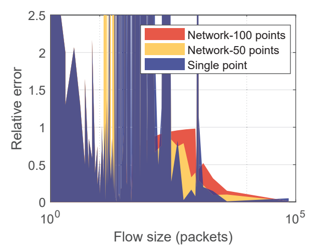
Accuracy vs. network scale (Fig. 20(b)- 20(d)). To fully increase the network scale and routing randomness, we consider extreme scenarios involving full-mesh topology with 50 and 100 measurement points. Each packet randomly selects an entry point and is subsequently randomly forwarded (with hop



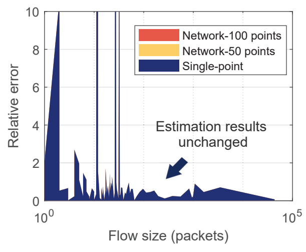
(a) Performance with memory usage.



(b) Performance of Jaqen with different flow size and network scale.

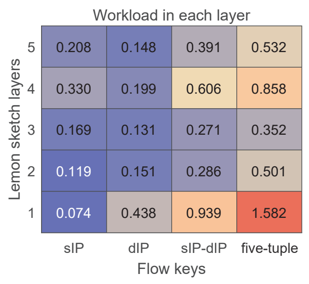


(c) Performance of Couper with different flow size and network scale.

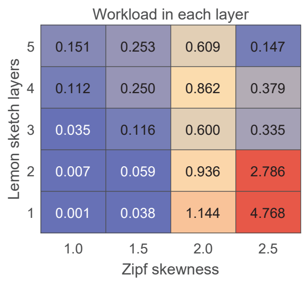


(d) Performance of Lemon with different flow size and network scale.

Figure 20: Performance with memory and network scale.



(a) Workload with flow key change.



(b) Workload with FSD change.

Figure 21: Lemon workload in each layer.

limit of 5). We investigate the accuracy changes of per-flow volume estimation (per five-tuple of 5M packets from MAWI). These large-scale networks exacerbate the over-counting issue in Jaqen and the mis-allocating in Couper. In contrast, Lemon employs a globally consistent update mechanism, ensuring that the error in per-flow estimation remains unaffected.

Workload across layers (Fig.21). We define the workload of $layer_i$ as the ratio of the number of flows estimated in (or higher than) this layer to the number of units S_i^s. Fig.21 shows that lower layers of Lemon experience more collision due to the dense distribution of smaller flows. Nevertheless, higher layers remain relatively unaffected, effectively handling the larger flows with minimal collision. In practical deployments, we recommend increasing the number of units S_i^s of the layer with a higher workload.

Resource	Percent
Meter ALU	14.58%
Hash Bit	4.51%
TCAM	0.35%
SRAM	10.31%

Table 2: Resources usage in programmable switch.

6.5 Hardware Resource Consumption

Table 2 shows the hardware resource usage with the hardware version of Lemon. Lemon sketch can meet the hardware resource constraints of programmable switches that only occupy 10.31% of total SRAM, and 0.35% TCAM. There are still adequate resources for other network functions.

7 Discussion

Duplicate packets detection. Lemon addresses the over-counting issue by leveraging unique packet identifiers derived from packet headers. However, potential attackers might attempt to evade detection by mixing duplicate packets with the same packet identifiers into DDoS traffic. To address this issue, we employ an approach by deploying a Count-Min (CM) sketch for per-flow packet counting alongside the Lemon sketch in the data plane as shown in Fig.22. Given Lemon's minimal resource usage (Section 6.5), deploying additional data structures concurrently is feasible. For a flow with key_{flow}, we will obtain two estimated results from the data plane at measurement point i, where E_i is the results from the Lemon sketch, representing the cardinality of different key_{pkg} in the flow; and C_i is the result from the CM sketch, representing the number of all packets. The network-wide aggregate result E_{net} of all E_i can be obtained from the global Lemon sketch. According to the method proposed in [20], we reconstruct final estimating result E'_{net} as:

$$E'_{net} = \frac{E_{net}}{n} \cdot \sum_{i=1}^{n} \frac{C_i}{E_i} \tag{10}$$

where n is the number of measurement points. When each packet has a unique key_{pkg}, $E_{net} \approx E'_{net}$. In the case of duplicate packets in the network, $E_{net} < E'_{net}$; then taking E'_{net} as the estimate will be more accurate.

Enhance existing mechanisms. As a fundamental measurement framework, Lemon's flexible measurement task configuration enables the collection of necessary network features for existing mechanisms, thereby supporting more complex detection and mitigation strategies.

- Patronum [57] measures entropy changes (detect attacks from botnets) and volume changes (detect heavy hitters). Such mechanism can be implemented by combining the volume-based and entropy-based methods by Lemon, giving Patronum network-wide detection capabilities.
- For DDoS attack detection based on machine learning or large language models [42], Lemon can collect per-flow level features as the input of such mechanisms.

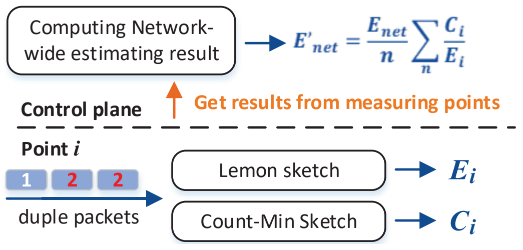

Computing Network-wide estimating result
$E'_{net} = \frac{E_{net}}{n}\sum_{n}\frac{C_i}{E_i}$
Get results from measuring points
Control plane
Point i
1
2
2
duple packets
Lemon sketch
Count-Min Sketch
E_i
C_i

Figure 22: Handling duplicate packets in the network-wide.

8 Related Works

Sketch-based DDoS detection. Existing measurement systems extensively utilize sketches (e.g., CM-Sketch [17], UnivMon [45], Elastic [60], CoCoSketch [63]) in the data plane to obtain network status efficiently. Sonata [30] introduced an architecture that leverages control plane primitives to flexibly deploy network measurement tasks alongside sketches in the data plane. Following Sonata, numerous subsequent studies have enhanced the data plane and control flow [39,45,46,48,64]. Among them, UnivMon [45] explores the use of sketch-based measurement systems in the context of network-wide measurements. Jaqen [46] leverages UnivMon as the data plane to build a DDoS attack detection system for ISP networks with multiple measurement points. Nevertheless, it assumes that the routing of packets is a prior knowledge to avoid packet over-counting issue. This assumption indeed does not hold in practice as we have analyzed.

Routing-oblivious measurement. Some studies have proposed routing-oblivious measurement methods to adapt to frequent routing changes. These works use hash-based probabilistic data structures that map identical packets to the same hash value across different measurement points. AROMA and SAROS [8,40] utilizes a k-partition hash structure, and INVEST [20] employs HyperLogLog for network-wide volume estimation. While these solutions mitigate the over-counting issue, they are primarily designed for specific measurement tasks (*e.g.,* heavy-hitter detection) and cannot provide fine-grained, per-flow measurements as sketch-based systems do. For instance, AROMA and SAROS can only estimate the volume of heavy-hitters, while INVEST only estimates the throughput of the entire network, and does not provide further information at a per-flow level. For per-flow level measurements, Beaucoup [15] reports flows exceeding a predefined threshold by allocating bitmaps to each flow (*i.e.,* couper collector), but cannot estimate the exact flow size, which is important for DDoS detection. Couper [52] performs per-flow cardinality estimation but suffers from stage mis-allocating issue, as illustrated in Section 2.2. In contrast, Lemon achieves accurate routing-oblivious per-flow measurement for network-wide DDoS detection.

9 Conclusion

Network-wide DDoS detection holds great promise for detecting and mitigating DDoS traffic upstream of convergence links. However, existing approaches have struggled with issues of packet over-counting and stage mis-allocating, which hinder effective network-wide deployment. To this end, we propose Lemon, a routing-oblivious, resource-efficient, and highly scalable network-wide DDoS detection system. Through extensive experiments and analyses, we show that Lemon outperforms existing solutions in network-wide DDoS attack detection and enables accurate per-flow measurements.

Acknowledgments

We thank all the reviewers and our shepherd for their insightful comments. This work is supported by National Key R&D Program of China (Grant No. 2022YFB3103000), by the National Natural Science Foundation of China (Grant No. U20A20180).

A Ethics Considerations

Our research does not involve any ethical issues.

B Open Science

We adhere to the open science policy. The Lemon prototype (both the Bmv2 version and hardware version) is publicly available [1, 3].

References

[1] Lemon. https://github.com/f-555/Lemon, 2024.

[2] Open-tofino. https://github.com/barefootnetworks/Open-Tofino, 2024.

[3] Lemon artifact for usenix 2025. https://doi.org/10.5281/zenodo.14729708, 2025.

[4] Y. Afek, A. Bremler-Barr, E. Cohen, S. L. Feibish, and M. Shagam. Efficient distinct heavy hitters for dns ddos attack detection. *arXiv preprint arXiv:1612.02636*, 2016.

[5] M. B. Anley, A. Genovese, D. Agostinello, and V. Piuri. Robust ddos attack detection with adaptive transfer learning. *Computers & Security*, 144:103962, 2024.

[6] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*, pages 1093–1110, 2017.

[7] B. Augustin, T. Friedman, and R. Teixeira. Measuring load-balanced paths in the internet. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 149–160, 2007.

[8] R. B. Basat, X. Chen, G. Einziger, S. L. Feibish, D. Raz, and M. Yu. Routing oblivious measurement analytics. In *2020 IFIP Networking Conference (Networking)*, pages 449–457. IEEE, 2020.

[9] R. B. Basat, G. Einziger, S. L. Feibish, J. Moraney, and D. Raz. Network-wide routing-oblivious heavy hitters. In *Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems*, pages 66–73, 2018.

[10] R. Ben-Basat, G. Einziger, S. L. Feibish, J. Moraney, B. Tayh, and D. Raz. Routing-oblivious network-wide measurements. *IEEE/ACM Transactions on Networking*, 29(6):2386–2398, 2021.

[11] B. Bencsáth and I. Vajda. Protection against ddos attacks based on traffic level measurements. In *2004 International Symposium on Collaborative Technologies and Systems*, pages 22–28. Citeseer, 2004.

[12] H. Birge-Lee, S. Yoo, B. Herber, J. Rexford, and M. Apostolaki. {TANGO}: Secure collaborative route control across the public internet. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1791–1811, 2024.

[13] S. Bjarnason. Ddos defences in the terabit era: Attack trends, carpet bombing. *APNIC blog, https://blog. apnic. net/2018/12/04/ddos-defences-in-the-terabit-era-attack-trends-carpet\-bombing*, 2018.

[14] J.-I. Castillo-Velazquez and N.-G. Velazquez-Cruz. Emulation of the updated canarie backbone network topology under ipv6 up to 2022. In *Proceedings of CECNet 2022*, pages 465–471. IOS Press, 2022.

[15] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford. Beaucoup: Answering many network traffic queries, one memory update at a time. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 226–239, 2020.

[16] K. Cho, K. Mitsuya, and A. Kato. Traffic data repository at the {WIDE} project. In *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, 2000.

[17] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[18] A. da Silveira Ilha, Â. C. Lapolli, J. A. Marques, and L. P. Gaspary. Euclid: A fully in-network, p4-based approach for real-time ddos attack detection and mitigation. *IEEE Transactions on Network and Service Management*, 18(3):3121–3139, 2020.

[19] D. Ding, M. Savi, F. Pederzolli, M. Campanella, and D. Siracusa. In-network volumetric ddos victim identification using programmable commodity switches. *IEEE Transactions on Network and Service Management*, 18(2):1191–1202, 2021.

[20] D. Ding, M. Savi, F. Pederzolli, and D. Siracusa. Invest: Flow-based traffic volume estimation in data-plane programmable networks. In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–9. IEEE, 2021.

[21] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM transactions on networking*, 9(3):280–292, 2001.

[22] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. Bohatei: Flexible and elastic {DDoS} defense. In *24th USENIX security symposium (USENIX Security 15)*, pages 817–832, 2015.

[23] S. L. Feibish, Y. Afek, A. Bremler-Barr, E. Cohen, and M. Shagam. Mitigating dns random subdomain ddos attacks by distinct heavy hitters sketches. In *Proceedings of the fifth ACM/IEEE workshop on hot topics in web systems and technologies*, pages 1–6, 2017.

[24] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, (Proceedings), 2007.

[25] G. Folino and P. Sabatino. Ensemble based collaborative and distributed intrusion detection systems: A survey. *Journal of Network and Computer Applications*, 66:1–16, 2016.

[26] S. Fortunati, F. Gini, M. S. Greco, A. Farina, A. Graziano, and S. Giompapa. An improvement of the state-of-the-art covariance-based methods for statistical anomaly detection algorithms. *Signal, Image and Video Processing*, 10:687–694, 2016.

[27] Y. Gao and Z. Wang. A review of p4 programmable data planes for network security. *Mobile Information Systems*, 2021(1):1257046, 2021.

[28] C. Goutte and E. Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European conference on information retrieval*, pages 345–359. Springer, 2005.

[29] F. Gui, S. Wang, D. Li, L. Chen, K. Gao, C. Min, and Y. Wang. Redte: Mitigating subsecond traffic bursts with real-time and distributed traffic engineering. In *Proceedings of the ACM SIGCOMM 2024 Conference*, pages 71–85, 2024.

[30] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger. Sonata: Query-driven streaming network telemetry. In *Proceedings of the 2018 conference of the ACM special interest group on data communication*, pages 357–371, 2018.

[31] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. *ACM SIGCOMM computer communication review*, 45(4):15–28, 2015.

[32] T. Heinrich, R. R. Obelheiro, and C. A. Maziero. New kids on the drdos block: Characterizing multiprotocol and carpet bombing attacks. In *International Conference on Passive and Active Network Measurement*, pages 269–283. Springer, 2021.

[33] S. Hizal, U. Cavusoglu, and D. Akgun. A novel deep learning-based intrusion detection system for iot ddos security. *Internet of Things*, 28:101336, 2024.

[34] Q. Huang, H. Sun, P. P. Lee, W. Bai, F. Zhu, and Y. Bao. Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 404–421, 2020.

[35] S. Kaur, K. Kumar, N. Aggarwal, and G. Singh. A comprehensive survey of ddos defense solutions in sdn: Taxonomy, research challenges, and future directions. *Computers & Security*, 110:102423, 2021.

[36] S. Kim, C. Jung, R. Jang, D. Mohaisen, and D. H. Nyang. A robust counting sketch for data plane intrusion detection. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023*. The Internet Society, 2023.

[37] A. M. Koster and M. Kutschka. Network design under demand uncertainties: A case study on the abilene and geant network data. In *Photonic Networks, 12. ITG Symposium*, pages 1–8. VDE, 2011.

[38] A. Kumar, M. Sung, J. Xu, and J. Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. *ACM SIGMETRICS Performance Evaluation Review*, 32(1):177–188, 2004.

[39] P. Laffranchini, L. Rodrigues, M. Canini, and B. Krishnamurthy. Measurements as first-class artifacts. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 415–423. IEEE, 2019.

[40] E. Li, W. Wu, W. Zhaohua, Z. Li, and J. Niu. Saros: A self-adaptive routing oblivious sampling method for network-wide heavy hitter detection. In *Proceedings of the 8th Asia-Pacific Workshop on Networking*, pages 142–148, 2024.

[41] Q. Li, H. Huang, R. Li, J. Lv, Z. Yuan, L. Ma, Y. Han, and Y. Jiang. A comprehensive survey on ddos defense systems: New trends and challenges. *Computer Networks*, page 109895, 2023.

[42] Q. Li, Y. Zhang, Z. Jia, Y. Hu, L. Zhang, J. Zhang, Y. Xu, Y. Cui, Z. Guo, and X. Zhang. Dollm: How large language models understanding network flow data to detect carpet bombing ddos. *arXiv preprint arXiv:2405.07638*, 2024.

[43] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[44] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 334–350. 2019.

[45] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[46] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar. Jaqen: A {High-Performance}{Switch-Native} approach for detecting and mitigating volumetric {DDoS} attacks with programmable switches. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3829–3846, 2021.

[47] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE network*, 8(3):26–41, 1994.

[48] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 85–98, 2017.

[49] M. Nooribakhsh and M. Mollamotalebi. A review on statistical approaches for anomaly detection in ddos attacks. *Information Security Journal: A Global Perspective*, 29(3):118–133, 2020.

[50] H. Shao, X. Wang, Y. Lu, Y. Yu, S. Zheng, and Y. Zhao. Accessing cloud with disaggregated {Software-Defined} router. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 1–14, 2021.

[51] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford. Heavy-hitter detection entirely in the data plane. In *Proceedings of the Symposium on SDN Research*, pages 164–176, 2017.

[52] X. Song, J. Zheng, H. Qian, S. Zhao, H. Zhang, X. Pan, and G. Chen. Couper: Memory-efficient cardinality estimation under unbalanced distribution. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 2753–2765. IEEE, 2023.

[53] Y. Tao and S. Yu. Ddos attack detection at local area networks using information theoretical metrics. In *2013 12th IEEE international conference on trust, security and privacy in computing and communications*, pages 233–240. IEEE, 2013.

[54] R. Vishwakarma and A. K. Jain. A survey of ddos attacking techniques and defence mechanisms in the iot network. *Telecommunication systems*, 73(1):3–25, 2020.

[55] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems (TODS)*, 15(2):208–229, 1990.

[56] B. Wu, K. Qian, B. Li, Y. Ma, Q. Zhang, Z. Jiang, J. Zhao, D. Cai, E. Zhai, X. Liu, et al. Xron: A hybrid elastic cloud overlay network for video conferencing at planetary scale. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 696–709, 2023.

[57] J. Wu, H. Pan, P. Cui, Y. Huang, J. Zhou, P. He, Y. Li, Z. Li, and G. Xie. Patronum: In-network volumetric ddos detection and mitigation with programmable switches. In *European Symposium on Research in Computer Security*, pages 187–207. Springer, 2024.

[58] K. Yang, S. Long, Q. Shi, Y. Li, Z. Liu, Y. Wu, T. Yang, and Z. Jia. Sketchint: Empowering int with towersketch for per-flow per-switch measurement. *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[59] K. Yang, Y. Wu, R. Miao, T. Yang, Z. Liu, Z. Xu, R. Qiu, Y. Zhao, H. Lv, Z. Ji, et al. Chamelemon: Shifting measurement attention as network state changes. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 881–903, 2023.

[60] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.

[61] X. Yuan, C. Li, and X. Li. Deepdefense: identifying ddos attack via deep learning. In *2017 IEEE international conference on smart computing (SMARTCOMP)*, pages 1–8. IEEE, 2017.

[62] 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 *the 27th Network and Distributed System Security Symposium (NDSS 2020)*, 2020.

[63] Y. Zhang, Z. Liu, R. Wang, T. Yang, J. Li, R. Miao, P. Liu, R. Zhang, and J. Jiang. Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 207–222, 2021.

[64] H. Zheng, C. Tian, T. Yang, H. Lin, C. Liu, Z. Zhang, W. Dou, and G. Chen. Flymon: enabling on-the-fly task reconfiguration for network measurement. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 486–502, 2022.

[65] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao, et al. Packet-level telemetry in large datacenter networks. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 479–491, 2015.