Accurate Traffic Measurement with Hierarchical Filtering

Haibo Wang, Hongli Xu, *Member, IEEE*, Liusheng Huang, *Member, IEEE*, Yutong Zhai

Abstract

Sketches have been widely used to record traffic statistics using sub-linear space data structure. Most sketches focus on the traffic estimation of elephant flows due to their significance to many network optimization tasks, *e.g.*, traffic engineering and load balancing. In fact, the information of aggregate mice flows (*e.g.*, all the mice flows with the same source IP) is also crucial to many security-associated tasks, *e.g.*, DDoS detection and network scan. However, the previous solutions, *e.g.*, measuring each individual flow or using multiple independent sketches for tasks, will result in worse estimation error or higher computational overhead. To conquer the above disadvantages, we propose an accurate traffic measurement framework with multiple filters, called **Sketchtree**, to efficiently measure both elephant flows and aggregate mice flows. These filters in Sketchtree are organized with a hierarchical manner, and help to alleviate the hash collision and improve the measurement accuracy, as the number of flows through hierarchical filters one by one will be decreased gradually. We also design some mechanisms to improve the resource efficiency. To validate our proposal, we have implemented Sketchtree with C, and conducted evaluation using real campus traffic traces. The experimental results show that Sketchtree can reduce the measurement error of aggregate mice flows by 60% on average, and preserve high measurement accuracy of elephant flows and processing speed, compared with state-of-the-art sketches.

Index Terms

Network Measurement; Hierarchical Filtering; Sketch; Attribute.

H. Wang, H. Xu (Corresponding Author), L. Huang and Y. Zhai are with the School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027, and also with Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, Jiangsu, China, 215123. E-mail: wanghaib@mail.ustc.edu.cn, xuhongli@ustc.edu.cn, lshuang@ustc.edu.cn, zyt1996@mail.ustc.edu.cn.

1. Introduction

Network measurement, with its goal to estimate the traffic size of network flows, is crucial to helping network operators make better network management decisions. To pursue better measurement accuracy, different solutions, *e.g.*, using flow tables and packet sampling, have been proposed. However, due to the limited resources (*e.g.*, TCAM and CPU computing capacity) on switches, these solutions often face with some disadvantages. For example, SDN switches can measure the traffic size using flow tables (or flow entries). Since the number of flows (*e.g.*, 10^6 [1]) usually far exceeds the flow table size (*e.g.*, 16K on most switches), it is impossible to measure the traffic size of each individual flow only using the flow tables. Meanwhile, packet sampling measures flow information with a predefined sampling rate, *e.g.*, 0.01. However, it suffers from low measurement accuracy as only partial flows are sampled and some mice flows may be discarded. Although the measurement accuracy can be improved by increasing the sampling rate or even recording all the traffic, *e.g.*, SPAN [2], the required resources will dramatically increase and pose scalability issues especially in high-speed networks.

On the contrary, sketches can provide an alternative solution to achieve fine-grained traffic measurement with compact data structures, which summarize traffic statistics of all packets with fixed-size memory. Due to resource constraints on switches, most sketches focus on the statistics of elephant flows, as these flows are usually relevant in many network optimization tasks. For example, heavy hitter detection identifies large flows whose byte volumes are above a threshold [3], while load balancing can be achieved by redirecting some elephant flows [4].

Aside from elephant flows, some statistics information of mice flows is of equal importance in many applications. In particular, some security-related tasks require the traffic information of aggregate mice flows with different attributes (*e.g.*, all the mice flows with the same source IP prefix) [5], regardless of the information of each individual mice flow. For example, DDoS attack refers that a number of sources send packets to a restricted destination [6]. To detect this attack, we need the overall traffic amount of (mice) flows to a destination. For convenience, we call these mice flows aggregated with the attribute of destination address. Another example is network scan [7], the opposite of DDoS. It requires the overall traffic of mice flows from a specific source (*i.e.*, aggregate mice flows with the attribute of source address).

Therefore, traffic measurements of both elephant flows and aggregate mice flows are important and necessary. One may think that it is easy to infer the statistics of aggregate mice flows with the information of each individual (mice) flow. However, this solution poses two challenges. (1) Though many sketches can work well for elephant flows, they often can not guarantee high measurement accuracy for each individual mice flow. For instance, our experimental results on some sketches (*e.g.*, Count-Min [8], Cold Filter [9]) show that the average measurement error of mice flows is several times of their real frequency. Especially, when the mice flow is misreported as an elephant flow, the estimation error will be much larger. Thus, the sum of all individual flows in an aggregate flow may violate its real frequency a lot (results are shown in Section 6.4). (2) Sometimes it is difficult to figure out all the individual flows in an aggregate mice flow. For example, to estimate an aggregate flow to a certain destination in DDoS attack, we need to find out all the sources in the network, which is time and resource consuming.

Another way for traffic measurement of both elephant flows and aggregate mice flows is adopting one independent sketch for each specific measurement task. For example, elephant flows can be recorded by Count-Min, and an additional Count-Min sketch is applied to record the traffic information of aggregate mice flows with the attribute of destination address. Note that these two Count-Min sketches are mutual-independent, as they hash different parts of packet header into distinct data structures. Moreover, if there are multiple tasks, *e.g.*, DDoS and network scan, the aggregate mice flows ought to be recorded with different attributes. As a result, multiple independent sketches are required. However, this solution requires each sketch to independently process all the packets in the data stream, which is unaffordable for switches as their computational resources are limited, especially when there are multiple tasks.

To this end, we propose an accurate traffic measurement framework for both elephant flows and aggregate mice flows. This framework comprises multiple filters, each associated with a specific task, in which each filter will send the task-related flows to the corresponding sketch and others to the next filter for further measurement. Since this multi-filter framework looks like a binary tree, we call it **Sketchtree**. Sketchtree is task-oriented and adaptive to different tasks. Specifically, aggregate mice flows are measured with different attributes, which can be dynamically configured according to the different application requirements. Our main contributions can be summarized as follows:

- We inherit the idea of filtering, and apply multiple filters into Sketchtree. Specifically, Sketchtree uses the first filter to separate elephant and mice flows, and other filters to separate aggregate mice flows with different attributes. These filters in Sketchtree are organized with a hierarchical manner, which helps to gradually decrease the number of flows through hierarchical filters one by one, and further reduces the probability of hash collision in traffic measurement. Thus, Sketchtree can achieve high measurement accuracy of both elephant flows and aggregate mice flows.
- To decrease the memory and computation consumption, we design some optimization methods, such as building Huffman tree [10] by adjusting the orders of attributes and separating hash values. With these methods, our solution can achieve computation/memory efficiency.
- We have implemented Sketchtree with C. The experimental results on real traffic traces show Sketchtree can largely improve the measurement accuracy of aggregate mice flows, while preserving the high measurement accuracy of elephant flows and high processing speed, compared with state-of-the-art methods like Count-Min [8] and Cold Filter [9], under the same memory size.

2. Background and Motivation

This section first introduces some typical sketches, and then presents the challenges.

A. Related Works and Typical Sketches

Network measurement has been extensively studied in the context of different techniques such as sampling [11], wavelets [12], [13], and sketches [8], [14], [15], [16], [17]. Among these techniques, sketches are widely used for its high processing speed, low memory consumption and high measurement accuracy for (elephant) flows. The most acknowledged sketch is Count-Min

(CM) [8], which consists of d arrays, each associated with a hash function, $h_i(\cdot)_{1\leq i\leq d}$. For each incoming flow f with frequency N_f, Count-Min hashes flow f to d counters $h_i(f)_{1\leq i\leq d}$ using d hash functions, and increments the values of all the d hashed counters $value(h_i(f))_{1\leq i\leq d}$ by N_f. When querying the estimated frequency of this flow, Count-Min reports the minimum value among these d hashed counters, *i.e.*, $\min_{1\leq i\leq d}\{value(h_i(f)\}$. Count-Min keeps approximate counts for all flows and reports non-negative results compared with the real frequency. To reduce the estimated error, another sketch, called CM-CU [18], only increases the frequency of counter(s) whose values are smallest among d hashed counters.

Some sketches, based on CM and CM-CU, support top-k queries with addition memory, *e.g.*, a heap [15], a hierarchical data structure [19], or an array with k counters [20], [21]. These sketches can achieve high precessing speed when the number of elephant flows is small, *e.g.*, 32 in ASketch [21]. What's more, the literatures about sketches for specifically recording aggregate mice flows are rare but the idea about estimating the aggregate flows is not new. For instance, individual flows can be aggregated by a certain matching rule (also called attribute), *e.g.*, source or destination address, and then recorded by current counter-based sketches and Top-k sketches [16]. The survey of more sketches can be found in [22].

Instead of processing data stream in one sketch, recent research called Cold Filter [9] applies a filter to separate high-frequency flows from data stream at the first stage and then sends these high-frequency flows to the second stage (aforementioned common sketches) for further measurement. Specifically, Cold Filter uses a two-layer sketch with small-size counters to record the frequencies of mice flows. The second layer is counted only when the hashed counters in the first layer overflow. If all the hashed counters overflow at both layers, Cold Filter will report the incoming packet as an elephant flow and send it to the existing sketches, *e.g.*, CM-CU [18] and Space-Saving [20]. However, Cold Filter aims to improve the measurement accuracy of individual flows (especially elephant flows), rather than aggregate mice flows.

B. Information of Elephant Flows and Aggregate Mice Flows Matters

To maintain qualified network performance, operators need to serve many applications, *e.g.*, load balancing and cyber security. These applications require traffic statistics of both elephant flows and aggregate mice flows. Most of the previous works pay great attention to elephant flows, such as heavy hitters. However, the traffic statistics of aggregate mice flows are sometimes of equal importance in some security-associated tasks, *e.g.*, traffic anomalies. Here, we list some important tasks that require the statistics of elephant flows or aggregate mice flows as follows:

- **Heavy Hitters** [3]: identify elephant flows that consume more than a threshold of link capacity during a time window.
- **Traffic changes detection** [23]: indicate flows which trigger the most traffic changes over two consecutive time windows. This task requires the information of elephant flows.
- **DDoS** [24]: [1] multiple hosts send more than a threshold of data to a specific destination host within a time window. DDoS attack can be detected using the traffic size of aggregate mice flows with the attribute of destination address.
- **Network scan** [7]: a source host sends more than a threshold of data to multiple destination hosts within a time window. This anomaly can be detected using the traffic size of aggregate mice flows with the attribute of source address.

The above tasks require accurate information of elephant flows or aggregate mice flows. For example, network scan requires traffic information of aggregate mice flows with the attribute of source address. That is, what matters is the overall traffic of mice flows from one source address, rather than the detailed traffic size of each individual mice flow. It is worth noting that the aggregation way of mice flows depends on the requirements of tasks. For a different task like DDoS, the traffic size of aggregate mice flows classified by the destination address is necessary. Due to the diversity of tasks, we need to measure aggregate mice flows with different attributes. Therefore, we need to measure both elephant flows and aggregate mice flows (with different attributes) accurately, which is the main contribution of this paper.

[1]Due to the diversity of attacking methods, *e.g.*, ICMP, LAND, IP, *etc.* [5], the DDoS detection can be conducted through traffic volume (also called the number of packets) or the count of source hosts. This paper adopts the traffic volume of aggregate mice flows like [5], [7].

C. Hash Collision

Typically, most sketches, *e.g.*, CM and CM-CU, use multiple hash functions to summarize massive data streams within a limited memory size. As a result, measurement error in these sketches is mainly caused by hash collision among different flows, which may lead to inaccurate measurement results. There are two types of traffic measurement errors.

- Error A: multiple mice flows collide with each other in the same counter. As a result, the sketch gives inaccurate results for mice flows, but won't misreport them as elephant flows.
- Error B: A mice flow collides with elephant flow(s). Under this circumstance, the sketch may misclassify a mice flow and report it as an elephant flow.

Note that Error B is more serious than Error A as the traffic size of an elephant flow is usually several magnitudes as that of a mice flow. To solve this problem, we inherit the filtering idea to separate the mice flows and elephant flows. Specifically, the filter pre-determines whether the flow is an elephant flow or not (a mice flow) with a predefined threshold. After separation, elephant flows and mice flows are processed independently by different sketches. Therefore, Error B happens with a small probability. What's more, Error A is caused from the hash collision among mice flows. Due to a huge number of mice flows in a network, Error A can hardly be avoided, even if the sketch consumes more resources, *e.g.*, computing resources and memory size. Combined with Section 2.2, what matters is not the traffic information of individual mice flows, but that of aggregate mice flows. Thus, we should measure aggregate mice flows to serve different tasks, which can largely reduce the probability of hash collision.

D. Heavy Computational Overhead

Current commodity switches are often equipped with limited computational capacity, which constraints the processing speed for different sketches. Let's take the Count-Min sketch as an example. The inserting speed for Count-Min is about 20M packets per second [9]. For TCP/IP protocol, its packet size is at most 1500B. To be more practical, the average packet size is usually much less than the maximum packet size, *e.g.*, $\frac{1}{3}$. That is, the throughput of the Count-Min sketch can be 20M$\times$500B = 80Gbps. Since modern data center network scales to 40 Gbps

sketch	CPU cycle	sketch	CPU cycle
Bloom Filter [27]	77	CountSketch [15]	175
Count-Min	78	FlowRadar [26]	2584
CM-CU	97	RevSketch [28]	3858
Cold Filter	102	Unimon [29]	4382
FSS [30]	120	Deltoid [31]	10454

TABLE I: Comparison on computational overhead of existing sketches

or even higher speed [25], the throughput of lightweight sketches like Count-Min is qualified but still in the same magnitude of the bandwidth. However, CPU resources will become scarcer for two reasons. (1) Some individual sketches with additional functions improve the measurement ability/accuracy, but also consume more CPU resources. For example, FlowRadar [26] can detect heavy changes by comparing the frequency of a flow in two adjacent time windows relying on an Invertible Bloom Lookup Table (LBLT) [26]. However, its computational overhead and throughput are 2584 CPU cycles per packet and 2.5Gbps [17], respectively. Some computational overheads of existing sketches are shown in Table I[2]. (2) More seriously, multiple sketches may be deployed on a switch to support different application requirements. Although some lightweight sketches, like CM and CM-CU, are able to handle the incoming packets in real time, the switch becomes heavy-loaded if running multiple sketches simultaneously. Thus, we expect that multiple tasks, *e.g.*, DDoS detection, network scan detection, are conducted by one ”big” but lightweight sketch, rather than multiple individual sketches, with each sketch handling one task. That’s part of our research motivation, *i.e.*, fast measuring elephant flows and aggregate mice flows with different attributes using one “big” sketch.

3. Sketchtree Overview

Sketchtree is a robust network measurement framework for elephant flows and aggregate mice flows. Since some tasks focus on the information of mice flows with different attributes, we need to measure these different aggregate mice flows in our framework. We should note that there are t attributes of aggregate mice flows. Specifically, t also means the number of security-associated tasks, *e.g.*, 2 (one is the attribute of source address for network scan, the other is the attribute of destination address for DDoS attack), and is usually limited.

[2]The computational overheads of FlowRadar, RevSketch, Unimon and Deltoid are cited from Sketchvisor [17], and the results of remaining sketches come from our experiments, the detailed experimental settings are presented in Section 6.2.

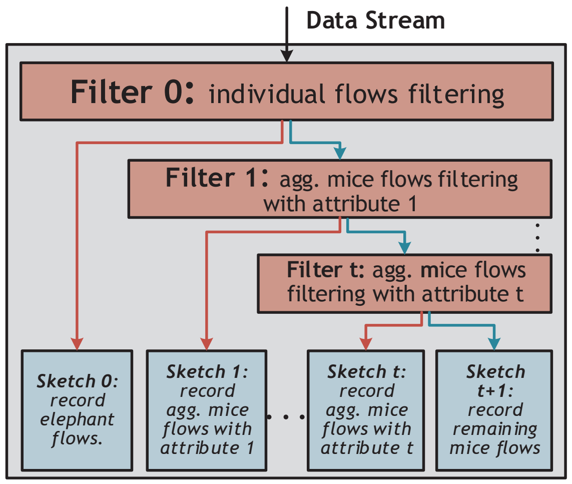

Data Stream
Filter 0: individual flows filtering
Filter 1: agg. mice flows filtering with attribute 1
Filter t: agg. mice flows filtering with attribute t
Sketch 0: record elephant flows.
Sketch 1: record agg. mice flows with attribute 1
Sketch t: record agg. mice flows with attribute t
Sketch t+1: record remaining mice flows

Fig. 1: Sketchtree Architecture

We observe that these different applications/tasks, including elephant-flow measurement and t security-associated tasks, usually require the information of mutually exclusive flows. On one hand, elephant-flow measurement only cares for those elephant flows, while security-associated tasks require the information derived from mice flows. On the other hand, for each security-associated task, it needs the information of aggregate mice flows whose overall frequencies are above a pre-defined threshold, denoted by LGM flows, because only these LGM flows can be potential attacks/dangers. Among different security-associated tasks, LGM flows with different attributes usually don't share individual mice flows, as a mice flow can hardly serve multiple security-associated tasks simultaneously [5] [32]. For example, a DDoS-associated flow may be sent to the destination host/server during Three-way Handshaking in TCP protocol, while network scan may be conducted after the completion of Three-way Handshaking. Thus, under most circumstances, these tasks need to measure different subsets of flows. Moreover, even if a mice flow serves multiple security-associated tasks, it is possible that detection results of these tasks are not affected. That's because the overall estimated frequency of an LGM flow still exceeds the threshold, despite that one or several individual mice flows are missed. To this end, we propose to gradually separate these subsets of flows using different filters one by one, so that each subset of filtered flows will be measured by different sketches in Sketchtree. As a result, the whole framework works like a ”big” sketch.

Figure 1 presents the architecture of Sketchtree, which consists of two stages: filtering stage and measurement stage. Generally, filtering stage includes $t + 1$ filters, namely, Filters 0-t, and measurement stage consists of $t + 2$ sketches, namely, Sketches 0 to $(t + 1)$. Each filter is responsible for flow separation and each sketch is responsible for flow measurement. Specifically, Filter i $(0 \leq i \leq t - 1)$ will send LGM flows (or elephant flows when $i = 0$) to Sketch i and remaining mice flows to Filter $i + 1$. The only exception is the last filter, namely, Filter t. It sends the remaining mice flows to sketch $t + 1$ for further measurement. Admittedly, as these mice flows may not be significant to the above t filters (or t tasks, as the types of attributes are usually associated with the requirement of different tasks) and sketch $t + 1$ can be removed in terms of the measurement of elephant flows and aggregate mice flows. It's worth noting that Sketch $t + 1$ is lightweight but necessary to some tasks, *e.g.*, flow size distribution, entropy estimation, as the traffic of these mice flows is much less than the total traffic. In this paper, we preserve Sketch $t + 1$ and the experimental results show that it can largely reduce the measurement accuracy of mice flows. We present the function of each sketch/filter as follows.

- **Elephant flows in Sketch 0**: All elephant flows will first be processed by Filter 0 and then sent to sketch 0 for further measurement. Thus, all the information of elephant flows is stored in Sketch 0.
- **Aggregate mice flows in Sketches/Filters 1 to** t: According to the types of attributes, these aggregate mice flows are recorded in different sketches. Each sketch is responsible for a distinct task, *e.g.*, DDoS. Apart from LGM flows, there are also aggregate mice flows whose frequencies are less than the threshold, which can be found at the associated filter. Since these tasks usually focus on LGM flows, the filter only sends LGM flows to the corresponding sketch for further measurement.
- **Individual Mice flows in Sketch** $t + 1$ **and Filter 0**: As a by-product of Sketchtree, the information of individual mice flows can be divided into two parts. The first part exists in Filter 0. In particular, Filter 0 records each individual flow and distinguishes whether the flow is large or small according to the estimated frequency. The filter can stop recording only when the estimated frequency of the flow exceeds the threshold. Since the estimated frequency of mice flows is not larger than the threshold, the measurement of mice flows

will not be affected. The other part is Sketch $t+1$, which records the information of mice flows that are swapped out by the above filters.

In order to promise effectiveness and robustness of a measurement framework, Sketchtree achieves high measurement accuracy, resource efficiency, and adaptivity to diversity of tasks.

Accuracy guarantee: these two different types of flows, *i.e.*, elephant flows and aggregate mice flows, are recorded independently, that is, different types of flows will not collide with each other. As explained in Section 2.3, this mutual-independent measurement method guarantees that hash collisions happen with a very small probability. For example, after filter 0, the number of flows has been drastically reduced in Sketch 0. Thus, an elephant flow is not likely to collide with other flows. Thus, the measurement accuracy can be guaranteed.

Computation efficiency: Sketchtree is responsible for multiple tasks. As discussed in Section 1, a typical solution for multiple tasks is to use multiple independent sketches, each in charge of a specific task. Compared with this solution, Sketchtree, can largely reduce the total computational overhead. Specifically, Sketchtree uses multiple filters to gradually classify the dataset into multiple subsets of flows, each corresponding to a specific task. For each task, Sketchtree further measures the task-related subset of flows, while the traditional solution needs to process all the packets in the dataset. Thus, Sketchtree can save the total computational consumption.

Adaptivity protect: there are t attributes, which may be diverse for different tasks. That is, these attributes are task-oriented. We can also change the orders of these attributes by changing the index of respective filters and sketches. Thus, Sketchtree is adaptive to diverse types of tasks, even with different priorities.

4. Our Solution

A. Algorithm Description

We give the explanations of some parameters for ease description.

- t: the number of attributes in Sketchtree.
- $\mathcal{T}_k$: the threshold setting in filter k. For example, $\mathcal{T}_0$ represents the threshold in Filter 0.

- $value_k(f)$: the estimated frequency of flow f in Filter k, with $0 \leq k \leq t$. For example, $value_0(f)$ means the estimated frequency of flow f in Filter 0.
- $\mathcal{F}_{k,f}$: the aggregate flow that flow f belongs to under attribute k.

Algorithm 1 Update process algorithm for Sketchtree

```
1:  Input: the incoming packet/flow $f$
2:  Record flow $f$ in Filter 0
3:  if $value_0(f) > \mathcal{T}_0$ then
4:      Send $f$ to Sketch 0
5:  else
6:      Update process for mice/aggregate flows
7:      $k = 1$ /*$k$ is the attribute index*/
8:      while $k \leq t$ do
9:          if $value_k(\mathcal{F}_{k,f}) > \mathcal{T}_k$ then
10:             Send flow $f$ to sketch $k$
11:             Sketch $k$ records the frequency of aggregate flow $\mathcal{F}_{k,f}$
12:             Break
13:         else
14:             if $k = t$ then
15:                 Send flow $f$ to Sketch t+1
16:                 Sketch $t$+1 records the frequency of flow $f$
17:         $k = k + 1$
18:         Aggregate flow $\mathcal{F}_{k,f}$ is recorded by Filter $k$
```

Update process of Sketchtree: Since Sketchtree records the frequency of a flow on the packet level, we consider a packet in the flow. Without loss of generality, we denote the packet in the flow by f as well. At first, the algorithm determines whether the flow is an elephant flow or a mice flow in Filter 0. If the estimated frequency of flow f is larger than a threshold, *i.e.*, $value_0(f) > \mathcal{T}_0$, this flow will be considered as an elephant flow and sent to sketch 0. Otherwise, flow f is a mice flow and sent to Filter 1, which records the frequency of each aggregate flow with attribute 1. If the aggregate flows are larger than the pre-defined threshold $\mathcal{T}_1$, the aggregate flow is considered large and sent to Sketch 1. Otherwise, it will be sent to the next filter. Each following filter works like filter 1 except the last filter, namely, filter t. If the aggregate flow in filter t is considered as small, the flow will be sent to sketch t+1. The algorithm is formally described in Alg. 1.

B. Filter Design and Sketch Selection

Strawman solution for filter design: one natural solution is to use a common sketch, *e.g.*, CM or CM-CU, as a filter. In particular, this sketch is used to record the frequency of each flow. For each incoming packet in this flow, we first query the estimated frequency. If the frequency is larger than the predefined threshold $\mathcal{T}_0$, this flow will be regarded as an elephant flow, otherwise, a mice flow. However, this method faces with two challenges. (1) Memory inefficiency. The size of each counter must be able to accommodate the threshold. Specifically, if the threshold $\mathcal{T}_0$ is 500, the counter's length should be 16 bits. Unfortunately, most of the flows are mice flows, which means most of the counters are not fully utilized, *i.e.*, most of counters only use a few bits to record mice flows [9]. (2) Computation inefficiency. All the hash functions are used for recording each packet. If the minimum value of the first one (or several) hashed counter(s) is less than the threshold, we can know the flow is a mice flow and the remaining hash operations can be saved.

Our filter design: similar to [9], we use two rows of counters, namely, Layer 1 and Layer 2, each consisting of w_1 and w_2 counters respectively. The sizes of each counter at Layers 1 and 2 are δ_1 and δ_2. For Layers 1 and 2, we allocate thresholds $\mathcal{T}'$ and $\mathcal{T}''$, with $\mathcal{T}_0 = \mathcal{T}' + \mathcal{T}''$, respectively. Note that Layer i ($i = 1, 2$) hashes each packet into h_i counters, and v_i ($i = 1, 2$) means the minimum value for h_i hashed counters in Layer i. For each incoming packet of a flow, (1) if $v_1 < \mathcal{T}'$, the filter increments the hashed counter(s) with the minimum value by 1 (the minimum value may locate at more than one counters), and the estimated frequency of the flow is $value_1 + 1$; (2) if $v_1 \geq \mathcal{T}'$, the filter records the flow in Layer 2. If $v_2 < \mathcal{T}''$, the filter increments the hashed counter(s) with the minimum value by 1 in Layer 2, and the estimated value is $\mathcal{T}' + v_2 + 1$. Otherwise, the flow overflows and will be regarded as an elephant flow.

Sketch selection: Each sketch records the filtered flows, including elephant flows and LGM flows. After filtering, the number of measured flows is largely reduced compared with the original dataset. Thus, estimating these flows with high accuracy is accessible. To further explore the high throughput of the sketch, we would like to select the sketch with light computational overhead, *e.g.*, CM, CM-CU, *etc.*. Here, we choose CM-CU to measure filtered flows for its higher accuracy than Count-Min under the same parameter setting. That is, Sketch $i_{0\leq i\leq t+1}$ is a CM-CU sketch. The detailed parameter settings for Sketch $i_{0\leq i\leq t+1}$ are shown in Section 6.2.

C. Two Optimization Methods for Resource Efficiency

To reduce the resource (*e.g.*, computation and memory) consumption on switches, we propose two optimization methods to make Sketchtree work more efficiently.

1) Organizing Sketchtree as a Huffman Tree (Optimization 1): As shown in Figure 1, Sketchtree can be recognized as a binary tree, where Filter 0 is the root of the tree. In particular, each filter is an internal node and each sketch is a leaf node. Take Filter 1 as an example. If the estimated frequency of a flow is larger than the threshold in filter 1, it will be sent to the left child node, *i.e.*, Sketch 1, otherwise, to the right child node, *i.e.*, filter 2. Therefore, the processing pipeline can be regarded as search in the binary tree. To reduce the computation consumption, we hope that the expected computational overhead for a packet (or a flow) should be minimized. That is, among all the sketches, we want the sketch that processes more packets to be placed with lower depth. Here, let's see the first left leaf node, namely Sketch 0. It processes all the elephant flows. Since the number of packets from the elephant flows are the majority of the total packets, the placement of Sketch 0 accords with our expectation to minimize the total computational overhead.

Let's consider the placement of Sketches 1 to t. Each parent node (or filter) records the aggregate flows with a distinct attribute, which associates with a specific task. According to the architecture of Sketchtree, Filters 1 to t process aggregate mice flows with attributes 1 to t and the orders of these attributes can be exchanged. As we have explained in Section 3, the LGB flows with different attributes usually do not share the same mice flows, thus different orders of these attributes will not affect the measurement results. However, since the numbers of LGB flows with different attributes are usually distinct, the orders of these attributes can influence the expected computational overhead for a packet in Sketchtree. To minimize the expected computational overhead of Sketchtree, we can exchange the placement orders of these attributes. As a result, the whole architecture becomes a Huffman tree. This adjustment can be conducted by the long-term flow management or during the system running to achieve the minimum expected computational overhead. Moreover, the stableness of Sketchtree can be maintained as we only need to exchange the indexes of filters and sketches.

2) Saving Memory and Computation Usage (Optimization 2) : **Memory-saving in the filtering stage**: each filter needs to distinguish elephant flows from the data stream rather than to estimate the frequency of elephant flows. In particular, if the value of a counter exceeds the predefined threshold $\mathcal{T}$, the flow will be regarded as an elephant flow and the filter does not need to update the values of hashed counters. Thus the size of a counter can be δ, with $2^{\delta} - 1 = \mathcal{T}$. For example, for the first layer in our filters, $\mathcal{T} = 15$ and $\delta = 4$. Under this circumstance, we only allocate 4 bits for each counter in Layer 1, instead of 32 bits, which is the normal size of a counter for the common sketches, *e.g.*, Count-Min [9]. As a result, $\frac{7}{8}$ of the memory size for Layer 1 is saved.

Memory-saving in the measurement stage: generally, the frequencies of most (aggregate) flows will not exceed the threshold, and only a minority of (aggregate) flows will be sent to the specific sketch for further measurement. For example, in practical traffic traces, elephant flows may be the top-1% flows in the data stream. That is, the number of flows measured by Sketch 0 will be much less than the number of all the flows in the dataset. Compared with the independent sketches (*e.g.*, Count-Min) which measure all the flows in the dataset, Sketch 0 achieve the same measurement accuracy of elephant flows with less counters per row. Thus, we can reduce the number of counters in each row to save memory usage.

Computation-saving: the previous works and our experiments show that most computational overhead lies in hash operations. Thus we expect to reduce hash computational overhead and our optimization method can locate h counters by one hash function. Specifically, we split the large-bit hash value (usually 32 bits [9]) into multiple segments, and each segment (or the combination of segments) is used to locate a counter. For example, for Layer 2 in the filter with the number of counters $w = 2^{16}$, $\delta = 16$ and $h = 2$ (memory usage is 0.13 MB), we split the 32-bit hash value into two 16-bit segments to locate two counters, respectively. What's more, if the 32-bit hash value is not big enough to be divided into h independent δ-bit segments, we will apply combinations of segments to locate the counters. For example, for Layer 1 in the filer with $w = 2^{18}$, $\delta = 4$ and $h = 3$ (memory usage is 0.13 MB), we separate the 32-bit hash value into four segments, one 11-bit segment and three 7-bit segments. Each 7-bit segment is combined with 11-bit segments to be a 18-bit value. This value is used to locate a counter.

5. Performance Analysis

The key part of Sketchtree is multiple filters, each responsible for filtering a subset of flows. As we have mentioned in Section 4.2, Sketchtree selects a previous sketch, *e.g.*, CM-CU, for traffic measurement. CM-CU is an improved sketch of Count-Min, and its measurement error is bounded according to the analysis of Count-Min [8]. Here, we do not repeat the analysis of CM-CU and this section mainly focuses on the performance of filter, *i.e.*, the misreport rate of each filter.

We consider a time window $[1, E]$, in which the data stream is denoted by $\Gamma = \{f_1, f_2, ..., f_n\}$ and contains n flows. Flow f with $f \in \Gamma$ includes N_f packets, that is, the frequency of flow f is N_f. In this time window, we construct Sketchtree for measurement. Before the formal analysis of Sketchtree, we first give the definition about frequency distribution of data stream Γ.

Definition 1: For each time point j in this time window, note that $I_k[j]$ is the subset of flows whose current real frequencies are not less than k. Formally, $I_k[j] = \{f | N_f[j] \geq k, k \in Z^+\}$. Also, let $\Delta_k[j]$ be $I_k[j] - I_{k+1}[j]$, which means the subset of flows whose current frequencies are exactly k.

According to Alg. 1 in Section 4, only the flows whose frequencies are larger than $\mathcal{T}$ will be identified as an elephant flow, and sent to the respective sketch. Thus, Sketchtree can only misreport some mice flows whose real frequencies are smaller than $\mathcal{T}$ as filter can only produce false-positive error. For simplicity, we say these mice flows are *misreported* and denoted by I_{mr}. Thus, the misreported rate P_{mr} can be defined as follows:

$$P_{mr} = \frac{I_{mr}}{I_1[E] - I_{\mathcal{T}}[E]} \tag{1}$$

We first adopt the theory of standard Bloom filter to derive the P_{mr} of CM-CU.

Standard Bloom filter: A standard Bloom filter [27] distinguishes whether a flow exists in a set or not. It consists of w-bit array associated with d hash functions. When a packet arrives, each hash function in the filter hashes the packet to a counter, and sets the counter to one. When querying a flow, if all the d hashed counters are one, it reports true; otherwise, false. Note that, the standard bloom filter only has false positive errors. Specifically, it may report true for a flow that isn't in the set, but never reports false for a flow that is actually in the set. Formally, given w, d and the number of processed flows n, the false positive rate P_{f_p} in the standard Bloom filter is expressed as [27]:

$$P_{f_p}(w,d,n) = \left[1 - \left(1 - \frac{1}{w}\right)^{nd}\right]^d \approx \left(1 - e^{-\frac{nd}{w}}\right)^d \tag{2}$$

Here we have the following lemma for function $\bar{P}_{f_p}(w,d,x)$:

Lemma 1: Function $\bar{P}_{f_p}(w,d,x) = \frac{1}{x}\sum_{i=0}^{x-1} P_{f_p}(w,d,i)$, with $\forall x \in Z^+$, is a monotonic increasing function of x.

We introduce a multi-layer Bloom filter to build the relation between standard Bloom filter and CM-CU. This multi-layer Bloom filter is an array of standard Bloom filters with the same w, d and hash functions. Note that λ is the number of layers in the multi-layer Bloom filter. Each Bloom filter is associated with *level* which is equal to its index in the array from 1 to λ. For each incoming packet, the multi-layer Bloom filter will check levels from 1 to λ. It stops when level $i, 1 \le i \le k$, reports false, and we will set the d hashed counters in level i to one. When querying the frequency of this flow, we will find the last level that reports true for this flow and report the order number of the level as the estimated frequency of this flow. Note that there are d_1 rows of counters in the CM-CU sketch, each row associated with w_1 counters. Moreover, the size of each counter is δ_1 bits. The multi-layer Bloom filter is equivalent to CM-CU if $w = w_1$, $d = d_1$ and $\lambda = 2^\delta - 1$. Thus we can analyze the misreport rate of each CM-CU-based filter through the multi-layer Bloom filter.

We consider an arbitrary time j during time window $[1, E]$. For the time point j, let $\hat{N}_f[j]$ be the current estimated frequency reported by the multi-layer Bloom filter.

Definition 2: For each time point j, note that $J_k[j]$ is the subset of flows whose current estimated frequencies are not less than k. Formally, $J_k[j] = \{f|\hat{N}_f[j] \geq k, k \in Z^+\}$.

Lemma 2: The subsets of flows, $I_k[j]$ and $J_k[j]$, have the following relation:

$$|I_k[j]| \leq |J_k[j]| \leq |I_k[j]| + \sum_{i=1}^{k-1} \left[(|I_i[j]| - |I_{i+1}[j]|) \cdot \prod_{u=1}^{i} \bar{P}_{f_p}(w, d, |J_{k-u}[j]|) \right] \tag{3}$$

where $\bar{P}_{f_p}(w,d,x) = \frac{1}{x}\sum_{i=0}^{x-1} P_{f_p}(w,d,i), \forall x \in Z^+$.

Proof: Since the multi-payer Bloom filter is equivalent to CM-CU, and Bloom filter reports no false-negative error, the lower bound $|I_k[j]| \leq |J_k[j]|$ can be easily observed.

The upper bound of $J_k[j]$ is related to function $\bar{P}_{f_p}(.)$, which is called *past positive value*. This function solves the following problem: given an initial standard Bloom filter with w and d, and x flows, with each flow including only one packet, for each incoming flow/packet, we set the hashed counters to one (also called true). After processing all the x distinct flows in the Bloom filter , how many flows are expected to be reported true by mistake? The answer is $\bar{P}_{f_p}(w,d,x) \cdot x$. Specifically, we consider the false positive rate of each flow. For the i^{th} incoming flow, its false positive rate equals to $P_{f_p}(w,d,i-1)$, for there are total $i-1$ flows inserted to the Bloom filter previously. Thus, the expected number of distinct flows that are reported true by mistake is $\sum\limits_{i=0}^{x-1} P_{f_p}(w,d,i) = \bar{P}_{f_p}(w,d,x) \cdot x$.

Due to the false positive rate, the estimated frequency of a flow may be larger than its real frequency. Similar to [9], the contribution to $J_k[j]$ derives from the following k parts: (1) flows in set $I_k[j]$; (2) flows in set $I_{k-1}[j] - I_k[j]$ and experiencing one or more false positives from level 1 to level $k-1$; (3) flows in set $I_{k-2}[j] - I_{k-1}[j]$ and experiencing two or more false positives from level 1 to level $k-1$;...; (k) flows in set $I_1[j] - I_2[j]$and experiencing $k-1$ false positives from level 1 to level $k-1$. For the first part, the contribution to $J_k[j]$ is $I_k[j]$; for the second part, since there are at most $J_{k-1}[j]$ flows in level $k-1$, the maximum probability of experiencing false positives is $\bar{P}_{f_p}(w,d,J_{k-1}[j])$; for the third part, the size of set is $|I_{k-2}[j]| - |I_{k-1}[j]|$, and the maximum probability of experiencing such false positives is $\bar{P}_{f_p}(w,d,J_{k-1}[j]) \cdot \bar{P}_{f_p}(w,d,J_{k-2}[j])$;...; for the k^{th} part, the size of set is $|I_1[j]| - |I_2[j]|$, and the maximum probability of experiencing such false positives is $\prod_{u=1}^{k-1} \bar{P}_{f_p}(w,d,J_{k-u}[j])$; Considering all the above k parts, the lemma holds. ■

Proof: Similar to [9], $P_{f_p}(w,d,n)$ in Eq. (2) is a monotonic increasing function of the variable n. For simplicity, $P_{f_p}(w,d,i)$ is denoted by sequence $\{a_i\}$. Thus, we have $a_i < a_{i+1}$. For $\forall k \in Z^+$, we have:

$$\begin{aligned}
&a_i < a_{i+1} \Longleftrightarrow \sum_{i=0}^{k-1} a_i < k \cdot a_k \\
&\Longleftrightarrow \sum_{i=0}^{k-1} a_i + \sum_{i=0}^{k-1} a_i < k \cdot a_k + \sum_{i=0}^{k-1} a_i \\
&\Longleftrightarrow (k+1)\sum_{i=1}^{k-1} a_i < k \sum_{i=0}^{k} a_i \Longleftrightarrow \frac{1}{k}\sum_{i=0}^{k-1} a_i < \frac{1}{k+1}\sum_{i=0}^{k} a_i \\
&\Longleftrightarrow \bar{P}_{f_p}(w,d,k) < \bar{P}_{f_p}(w,d,k+1)
\end{aligned} \tag{4}$$

The lemma holds. ■

As $\bar{P}_{f_p}(w,d,x)$ is a monotonic function of x, $|J_k[j]|$ can be bounded recursively by Lemma 1. For simplicity, $|J_k[j]|^L$ and $|J_k[j]|^U$ represent the lower and upper bounds of $|J_k[j]|$, respectively.

Bound of P_{mr} of Sketchtree: At first, we give a general bound of P_{mr} of Sketchtree:

Lemma 3: The misreport rate of Sketchtree is bounded by:

$$P_{mr} \le \frac{\sum_{k=1}^{\mathcal{T}} \left\{ \left[1 - \prod_{u=1}^{k} \left(1 - \prod_{i=1}^{\mathcal{T}-k+1} P_{fp}(w,d,|J_i[t_u]|^L) \right) \right] \cdot |\Delta_k[E]| \right\}}{|I_1[E]| - |I_{\mathcal{T}+1}[E]|} \tag{5}$$

where $t_u, 1 \le u \le k$ represents the incoming time point of the u^{th} packet in the flow that includes k packets. Obviously, t_u depends on the flow/packet distribution of appearance time.

Proof: For each misreported flow, its estimated frequency must exceed the threshold $\mathcal{T}$, thus we divide the P_{mr} into $\mathcal{T}$ parts: P_{mr}^1, P_{mr}^2,..., $P_{mr}^{\mathcal{T}}$, where P_{mr}^k denotes the misreported rate of flows whose real frequencies are k, namely $\Delta_k[E] = \{f | N_f = k\}$. Obviously, we have:

$$P_{mr} = \frac{\sum_{k=1}^{\mathcal{T}} (P_{mr}^k \cdot |\Delta_k[E]|)}{\sum_{k=1}^{\mathcal{T}} |\Delta_k[E]|} \tag{6}$$

Here we consider an arbitrary flow $\gamma \in \Delta_k[E]$, which appears in the time window $[1,E]$ k times. Note that $t_1,\ t_2, ..., t_k$ are the k appearance time points. If flow γ is misreported and its real frequency is k, the flow must experience $\mathcal{T} - k + 1$ false positives from level 1 to level $\mathcal{T}$. Note that $P_{mr}^{k,u}, 1 \le u \le k$, is the rate that misreport happens exactly in the time t_u, and we have:

$$P_{mr}^k = 1 - \prod_{u=1}^{k} (1 - P_{mr}^{k,u})(1 \le u \le k \le \mathcal{T}) \tag{7}$$

For each $P_{mr}^{k,u}$, since the misreport happens in time point t_u, the $\mathcal{T} - k + 1$ times of false positives happen before or equal to t_u. That is, t_u is the maximum time point among these appearance time points of $\mathcal{T} - k + 1$ false positives. Since the $P_{f_p}(w, d, x)$ is the monotonic increasing function of x, we have:

$$P_{mr}^{k,u} \leq \prod_{i=1}^{\mathcal{T}-k+1} P_{f_p}(w, d, |J_i[t_u]|) \tag{8}$$

Combining with Eqs. (6) and (7), The lemma holds. ■

Generally, P_{mr} is closely related to the distribution of appearance order of each packet $t_1, t_2, ...$ in a flow, such as Gaussian, Poisson and so on. For simplicity, we employ the random order model [33], [34] defined below, and other distribution models can be analyzed as well.

Definition 3 (Random order model): For each flow $f = \{p_1, ..., p_{N_f}\}$ with N_f packets, each packet is coming independently and uniformly at random.

Under the random order model, each flow f can be uniformly partitioned into k (or N_f) small data streams, each of which contains E/k flows with one packet. Thus time window can be partitioned into k intervals: $[1, E/k], [E/k + 1, 2E/k]$,...,$[\frac{(k-1)E}{k} + 1, E]$. Assume that the flow appears in the middle of each interval, that is, $t_1 = E/(2k)$, $t_2 = \frac{3E}{(2k)}$,..., $t_k = \frac{(2k-1)E}{(2k)}$. Therefore, we have :

$$|I_i[t_u]| = \frac{2u - 1}{2k} \cdot |I_i[E]|(1 \leq i \leq \mathcal{T}, 1 \leq q \leq k \leq \mathcal{T}) \tag{9}$$

According to Lemma 2, $|J_i[t_u]|$ is bounded by $|I_i[t_u]|$. Thus the misreport rate P_{mr} is bounded under the random order model.

6. Performance Evaluation

A. Experimental Setup

Traffic dataset: we use the collected real-world traffic traces in our campus as dataset. This dataset contains 28M packets and 0.73M flows, in which each flow is identified by its 5-tuple.

Measurement tasks: we adopt two security-associated tasks, *i.e.*, detection of network scan and DDoS. To serve these two security-associated tasks, Sketchtree cares for aggregate mice flows with attributes of source and destination addresses, *i.e.*, *t*=2. According to Optimization 1 in Section 4.3.1, the order of filters processing aggregate mice flows with different attributes can be adjusted to minimize the computational overhead of Sketchtree. At the beginning of system running, since flow distribution on each sketch of Sketchtree is unaccessible, we arbitrarily put the source address as the first attribute. Then we will study the impact of Optimizations 1 on the computational overhead of Sketchtree in Section 6.5.

B. Benchmarks and Parameter Settings

We adopt two categories of benchmarks. The first category of benchmarks, *e.g.*, Count-Min (CM) and CM-CU, measures individual flows and gives estimated frequencies of all these individual flows. Moreover, they report the frequency of each LGB flow by adding up frequencies of all the contained mice flows. The second category of benchmarks, *e.g.*, CF+CM-CU, CM(src)+CM(dst) and CM-CU(src)+CM-CU(dst), are independent sketches measuring different types of flows. For example, CF+CM-CU consists of two parts, CF (Cold Filter [9]) and a CM-CU sketch. These two parts are combined to measure individual flows, especially elephant flows. CM(src)+CM(dst) contains two independent Count-Min sketches that measure aggregate mice flows with the attributes of source and destination addresses, respectively. Likewise, CM-CU(src)+CM-CU(dst) includes two independent CM-CU sketches that record aggregate mice flows with the attributes of source and destination addresses, respectively. The detailed description of Sketchtree and these benchmarks is as follows.

Sketchtree: Sketchtree mainly focuses on the measurement of elephant flows and two kinds of aggregate mice flows. For ease of expression, we say that Sketches 0, 1, and 2 record **elephant flows, LGM1 flows and LGM2 flows**, respectively. As a by-product, we also evaluate the measurement accuracy of individual mice flows in Filter 0 and Sketch 4. Moreover, there are three filters, namely, filter 0, filter 1 and filter 2. Let M_{tree} be the memory size of Sketchtree, M_p be the total memory of Sketches 0-3, and M_f be the total memory of three filters. We have $M_{tree} = M_p + M_f$ and set $M_p : M_f = 9 : 11$ like [9]. Each filter is implemented by one or two layers to ensure measurement effectiveness, and each sketch is equipped with d arrays, each associated with w counters. Note that the size of each counter is δ bits. M is the memory size of the layer and h is the number of hash functions in each layer. For ease of description, $\mathcal{T}$

Filter	**Layer**	δ	$Threshold$	h	$M : M_f$	**Sketch**	d	δ	h	$M : M_p$
Filter 1	Layer 1	4	16	3	19:45	Sketch 0	2	32	2	24:55
	Layer 2	16	750	2	2:15	Sketch 1	1	16	2	12:110
Filter 2	Layer 1	4	16	3	4:45	Sketch 2	1	16	2	1:11
	Layer 2	16	750	2	2:15	Sketch 3	2	8	2	4:11
Filter 3	Layer 1	16	750	2	2:9	-				

TABLE II: Parameter settings for Sketchtree

represents the threshold of an arbitrary filter, *i.e.*, $\mathcal{T} \in \{\mathcal{T}_0, \mathcal{T}_1, \mathcal{T}_2\}$. Table II lists the parameter settings for each sketch/filter.

CM/CM-CU: both two sketches are used to measure individual flows and infer an LGM flow by adding up frequencies of all its included mice flows. To explore as high measurement accuracy as possible, each sketch is allocated 5 rows of counters, each associated with a hash function. The size of each counter is 32 bits [9]. The value of columns w depends on the memory size, which ranges from 1.4 MB to 2.8 MB [9].

CF+CM-CU: CF+CM-CU contains a filter named CF (Cold Filter [9]) and a CM-CU sketch. Specifically, CF separates elephant flows and mice flows, and only sends elephant flows to CM-CU for further measurement. CF+CM-CU is applied to record individual flows, especially elephant flows. Let M_{cf} be the memory size of CF, and M_t be the memory size of CM-CU. As recommended in [9], $M_{cf} : M_t = 9 : 10$. Specifically, CF includes two layers with the same parameter settings as filter 1 in Sketchtree. The only difference is that CF is set with threshold $\mathcal{T} = 256$ in the second layer [9]. CM-CU contains 3 rows of counters each with a hash function. The size of each counter is 32 bits.

CM(src)+CM(dst): CM(src)+CM(dst) contains two CM sketches, which estimate the frequency of aggregate mice flows with attributes of source and destination address, respectively. For ease of description, CM(src)+CM(dst) is abbreviated as CM(src) and CM(dst) when compared with Sketchtree in terms of measurement accuracy of LGM1 and LGM2 flows, respectively. Both CM(src) and CM(dst) include 3 rows of counters. The size of each counter is 32 bits [9]. For fairness, the total memory size of both CM(src)+CM(dst) is equal to that of other sketches.

CM-CU(src)+CM-CU(dst): CM-CU(src)+CM-CU(dst) includes two CM-CU sketches that record the frequency of aggregate mice flows with two attributes, *i.e.*, source and destination addresses, respectively. Moreover, CM-CU(src)+CM-CU(dst) is abbreviated as CM-CU(src) and CM-

CU(dst) when compared with Sketchtree in terms of measurement accuracy of LGM1 flows and LGM2 flows, respectively. The parameter settings are the same as those of CM(src)+CM(dst).

We have implemented Sketchtree with C. In our implementation, we find that the ability to process packets on Open vSwitch (OVS) [35] is constrained by software interrupts of the NIC driver, and the inserting speed is relatively trivial. Therefore, we build implementations of these sketches and directly feed them data stream, despite that we have successfully embedded the C-based sketch module into OVS. Also, the real traffic traces cannot be built by testbed as the source address and destination address are numerous. As a result, we execute the C procedure by reading the traffic file directly.

C. Metrics

Average absolute error (AAE) in frequency estimation: AAE is formulated as :$\frac{1}{|\Gamma|}\sum_{f\in\Gamma}|\hat{N}_f-N_f|$, where $\hat{N}_f$ is the estimated frequency of flow f, N_f is the real frequency of flow f, and Γ is the flow set. We calculate AAE by querying the absolute error of each flow.

Average relative error (ARE) in frequency fstimation: ARE is calculated by $\frac{1}{|\Gamma|}\sum_{f\in\Gamma}\frac{|\hat{N}_f-N_f|}{N_f}$. Like the query of AAE, we get ARE by querying the relative error of each flow.

Inserting speed and CPU cycles per packet: inserting speed means the ability to conduct mega inserting operations per second (Mops). Moreover, we estimate the average required CPU cycles per packet for each benchmark to show the computational overhead of each sketch.

The number of covered flows (packets) on each sketch of Sketchtree: the distribution of measured (also called covered) flows in four different sketches, namely, Sketch i, with $i = 0, 1, 2, 3$. The results will show the absolute number of covered flows and the ratio of the number of covered flows to the number of all the flows in the dataset. Moreover, we use the number of covered packets on each sketch to show the processing load for each sketch, and also to show the traffic distribution on these sketches.

D. Evaluation Results

Before giving formal experimental results, we present Figures 2 and 3 to show ARE of CM, CM-CU and Sketchtree for LGM1 and LGM2 flows. Specifically, ARE of CM and CM-CU

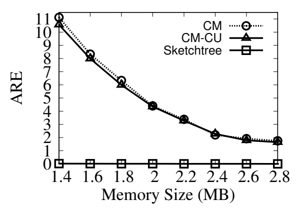


Fig. 2: ARE of LGM1 flows vs. Memory Size

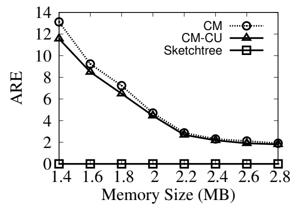


Fig. 3: ARE of LGM2 flows vs. Memory Size

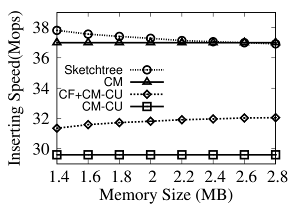


Fig. 4: Memory Size vs. Inserting Speed

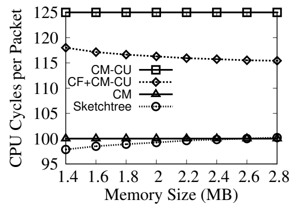
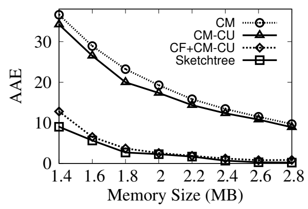
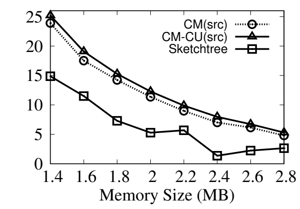
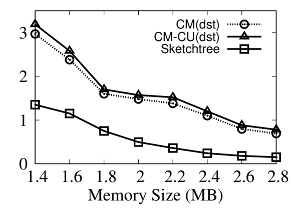


Fig. 5: CPU Cycles per Packet vs. Memory Size


(a) Elephant Flows


(b) LGM1 Flows


(c) LGM2 Flows

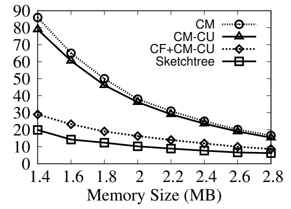
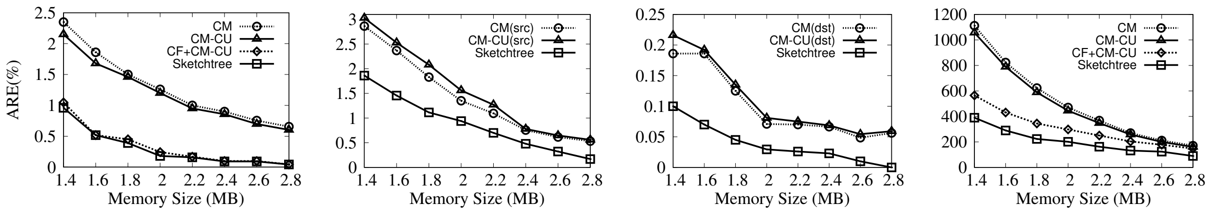

(d) Mice Flows

Fig. 6: AAE vs. Memory Size for Four Sketches

ranges from 2 to 14, while that of Sketchtree is less than 0.1. Thus it is impracticable to infer the frequency of an LGM flow by adding up frequencies of all its mice flows. In the following figures, we omit AAE and ARE of CM and CM-CU for LGM1 and LGM2 flows.

Inserting speed and CPU cycles per packet (Figures 4-5): we estimate the computational overhead for all sketches. As shown in Figures 4 and 5, Sketchtree leads to the fastest inserting speed, compared with other three benchmarks. Specifically, the inserting speeds of both S-ketchtree and CM achieve nearly 37Mops, while CF+CM-CU and CM-CU process less packets per second, reducing inserting speed by 5.5 Mops and 7.5 Mops, respectively. As for required CPU cycles per packet, Sketchtree shows its advantage as well, with the ability to insert a packet within 100 CPU cycles, while CM-CU and CF+CM-CU can only conduct one inserting operation with 115 and 125 CPU cycles, respectively.

AAE for elephant flows and aggregate mice flows (Figures 6(a)-6(c)): AAE of CM, CM-CU, CF+CM-CU and Sketchtree decreases as the memory size increases. In particular, (1) for the measurement of elephant flows, Sketchtree achieves the lowest AAE, which is only 80%, 11% and 13% of AAE by CF+CM-CU, CM and CM-CU, when the memory size is 2MB; (2) for the measurement of LGM1 flows, Sketchtree reduces AAE by 45% and 40% compared with CM(src) and CM-CU(src) when the memory size ranges from 1.4MB to 2.8MB; (3) for the measurement accuracy of LGM2 flows, Sketchtree dramatically outperforms CM(dst) and



(a) Elephant Flows (b) LGM1 Flows (c) LGM2 Flows (d) Mice Flows

Fig. 7: ARE vs. Memory Size for Four Sketches

CM-CU(dst), reducing AAE by nearly 60% compared with these two sketches.

ARE for elephant flows and aggregate mice flows (Figures 7(a)-7(c)): for the measurement accuracy of elephant flows, Figure 7(a) shows that Sketchtree achieves close ARE performance to CF+CM-CU, which is a state-of-the-art sketch for measurement of elephant flows. On the contrary, CM and CM-CU cannot record elephant flows precisely, increasing ARE by about 150% compared with CF+CM-CU and Sketchtree. As for the measurement accuracy of LGM1 flows in Figure 7(b), Sketchtree outperforms CM(src) and CM-CU(src), reducing ARE by averagely 50% and 46%, respectively. Moreover, compared with CM(dst) and CM-CU(dst), Sketchtree reduces ARE of LGM2 flows by about 70% and 67% by Figure 7(c), respectively.

AAE and ARE for mice flow measurement (Figures 6(d) and 7(d)): Figure 6(d) shows the AAE of these sketches for the measurement of mice flows. As we can see, CM and CM-CU lead to very large AAE, namely, 39 and 38, respectively, when the memory size is 2 MB. CF+CM-CU achieves only half of AAE compared with CM. What's more, Sketchtree reduces AAE by 33% and 80% compared with CF+CM-CU and CM, respectively. When the memory size exceeds 2 MB, only AAE of Sketchtree is less than 10. As for ARE in Figure 7(d), Sketchtree performs the best among these sketches, and ARE of Sketchtree is 88% when the memory size is 2.8 MB, which is very small. That's because Sketchtree not only separates the mice flows and elephant flows in filter 0, but also records part of the mice flows in the last sketch (Sketch 3). On the contrary, since there are a very large number of mice flows and the frequency of a mice flow is small, CF+CM-CU, CM and CM-CU are difficult to record mice flows precisely, increasing ARE by averagely 450%, 400% and 52% compared with Sketchtree, respectively.

Number of covered flows on each sketch of Sketchtree (Figure 8): as we can see, nearly 50% of flows, which are aggregated by source IP addresses, are processed by Sketch 1. Moreover,

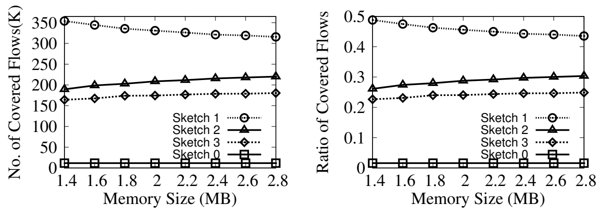


Fig. 8: Flow Distribution on Each Sketch of Sketchtree vs. Memory Size

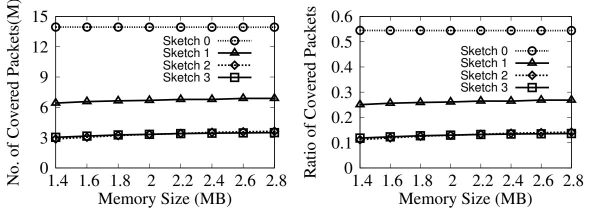
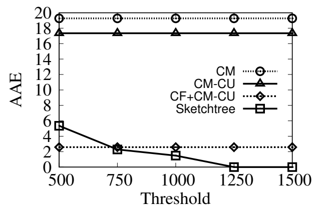


Fig. 9: Packet Distribution on Each Sketch of Sketchtree vs. Memory Size

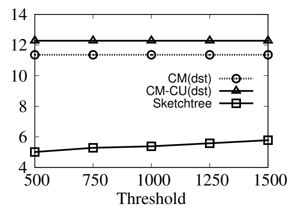
Sketch 2 and Sketch 3 process averagely 28% and 23% of flows, respectively, while Sketch 0 only records 1% of flows, which is verified by the fact that only 1% of flows are elephant flows in real dataset.

Number of covered packets on each sketch of Sketchtree (Figure 9): we observe distribution of packets processed by four sketches in Sketchtree. Figure 9 shows that a majority of packets are processed by Sketch 0. Combined with the results in Figure 8, Sketch 0 only records 1% of all flows, but the number of its processed packets takes up 55% of all packets by Figure 9, which shows that our dataset is heavy-tailed. Fortunately, the real traffic in practical networks also possesses this characteristic. Sketch 1 processes about 7M packets, which take up 25% of total packets. Other two sketches, Sketch 2 and Sketch 3, process the least number of packets, *i.e.*, 3.2M packets. By Figure 8, although Sketch 2 and Sketch 3 individually record over 22% of flows, they process a relatively small number of packets, which can reduce the average computational overhead per packet, for the number of packets in the bottom sketch is crucial to the overall computational overhead in Sketchtree.

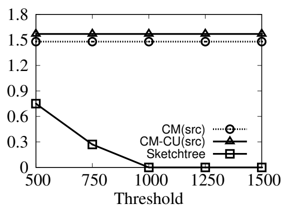
Summaries: (1) Sketchtree achieves the highest measurement accuracy for aggregate mice flows among four sketches, reducing measurement error (AAE and ARE) by nearly 50% compared with CM(src)+CM(dst) and CM-CU(src)+CM-CU(dst). (2) Sketchtree can record elephant flows accurately, reducing AAE (or ARE) by 10%-20% compared with CF+CM-CU, which is the state-of-the-art sketch for recording elephant flows. (3) Similar to some lightweight sketches, *e.g.*, CM, CM-CU, CF+CM-CU, Sketchtree maintains high processing speed. In all the experiments, Sketchtree consumes the least CPU cycles per packet among all sketches.



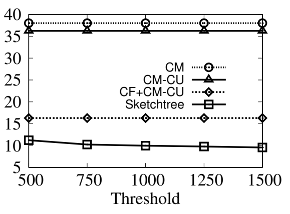
(a) Elephant Flows



(b) LGM1 Flows

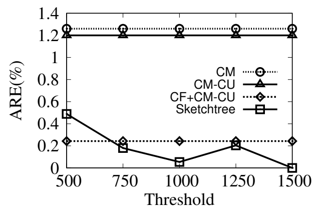


(c) LGM2 Flows

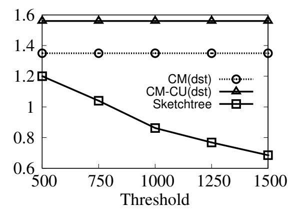


(d) Mice Flows

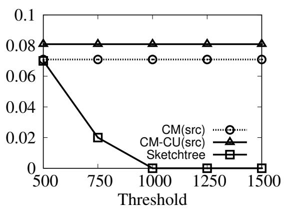
Fig. 10: AAE vs. Threshold $\mathcal{T}$ for Four Sketches



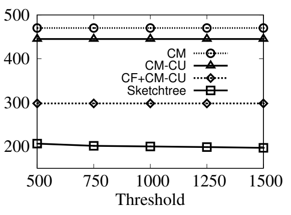
(a) Elephant Flows



(b) LGM1 Flows



(c) LGM2 Flows



(d) Mice Flows

Fig. 11: ARE vs. Threshold $\mathcal{T}$ for Four Sketches

E. Sensitivity Analysis

1) Sensitivity Analysis for Threshold $\mathcal{T}$: This part presents the impact of threshold $\mathcal{T}$ on different metrics. Other benchmarks are not affected by the threshold.

Impact on AAE (Figure 10): AAE for elephant flows, LGM2 flows and mice flows decreases while AEE for LGM1 flows increases, as the threshold increases. In particular, (1) for the measurement accuracy of elephant flows, when threshold exceeds 750, Sketchtree becomes the most accurate sketch. The reason is that when the threshold increases, less flows will be regarded as elephant flows. As a result, the probability of hash collisions in Sketch 0 decreases and the measurement accuracy decreases as well; (2) AAE for LGM2 flows drops quickly when the threshold increases from 500 to 1000; (3) for the measurement accuracy of LGM1 flows and mice flows, AAE of Sketchtree increases and decreases slowly as the threshold increases, respectively. For example, AAE of LGM1 flows increases by 1.8 when threshold increases from 500 to 1500.

Impact on ARE (Figure 11): Sketchtree outperforms CM, CM-CU and CF+CM-CU in terms of ARE, and ARE of Sketchtree decreases as the threshold increases. Specifically, (1) for the measurement accuracy of elephant flows, when the threshold increases to 750, the ARE of Sketchtree is lower than 0.2%, which is the least among all sketches; (2) as the threshold increases, ARE of LGM1 flows decreases. However, AAE of LGM1 flows increases by Figure

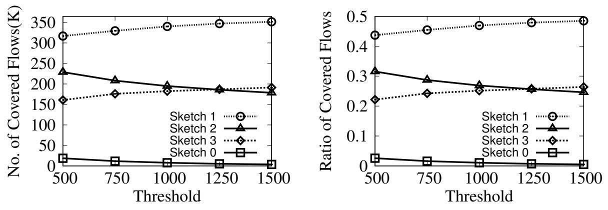


Fig. 12: Flow Distribution on Each Sketch of Sketchtree vs. Threshold $\mathcal{T}$

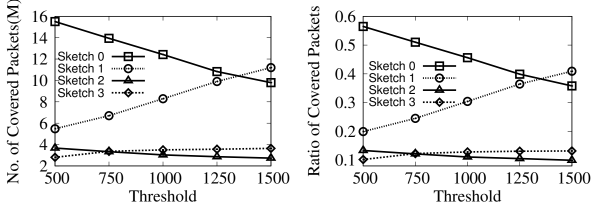


Fig. 13: Packet Distribution on Each Sketch of Sketchtree vs. Threshold $\mathcal{T}$

10(b). That's because the average frequency of LGM1 flows becomes larger, and therefore ARE becomes smaller. By comparison, (3) ARE of mice flows decreases as the threshold increases, which is the same trend as the AAE of mice flows. There are a number of mice flows in the network and the average frequency of mice flows is not drastically affected by the threshold. Thus both AAE and ARE of mice flows decrease as the threshold increases.

Impact on the number of covered flows on each sketch of Sketchtree (Figure 12): Figure 12 shows that the numbers of covered flows in Sketches 1 and 3 increase, while those in Sketches 0 and 2 decrease as the threshold increases. Specifically, Sketch 1 and Sketch 3 witness increases of 3.5K and 3K flows, respectively, when the threshold increases from 500 to 1500. Meanwhile, Sketch 2 losses 5K flows. Although Sketch 0 only decreases the number of covered flows by 1.5K as the threshold increases from 500 to 1500, it losses nearly 80% of its covered flows in Figure 12. As for the ratio of covered flows (right plot in Figure 12), Sketch 1 covers the most number of flows and Sketch 0 covers the least number of flows regardless of the increase of threshold. When the threshold is less than 1250, Sketch 2 covers more flows than Sketch 3. However, when the threshold is 1500, Sketch 3 covers more. That's because when the threshold is very large, more flows will be regarded as mice flows and processed by Sketch 3.

Impact on the number of covered packets on each sketch of Sketchtree (Figure 13): as we can see in Figure 13, the numbers of covered packets in Sketch 0 and Sketch 1 are very sensitive to the threshold, for the threshold directly determines whether the flow is regarded as an elephant flow or not. In particular, the number of covered packets in Sketch 0 decreases by nearly 6M as the threshold increases from 500 to 1500. These 6M flows are mainly supplemented to Sketch 1. What's more, Sketch 3 covers less than 4M packets, which is less than 15% of total packets. The reason is that when the threshold increases, although the number of its covered

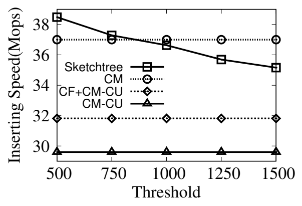


Fig. 14: Inserting Speed vs. Threshold $\mathcal{T}$

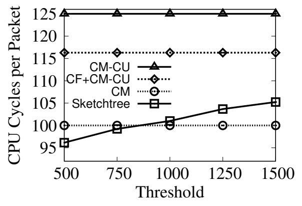


Fig. 15: CPU Cycles per Packet vs. Threshold $\mathcal{T}$

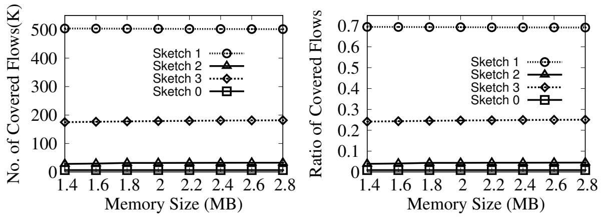


Fig. 16: Flow Distribution on Each Sketch of Sketchtree

flows increases, the absolute frequency of covered flows in Sketch 3 is not significant.

Impact on inserting speed and CPU cycles (Figures 14-15): as shown in Figure 14, a larger threshold decreases the inserting speed. Specifically, the inserting speed of Sketchtree decreases by 3.2 Mops when the threshold increases from 500 to 1500. That's because more flows are regarded as mice flows when the threshold becomes larger. These mice flows will be processed by more filters. As a result, the computational overhead is increased and inserting speed of Sketchtree is decreased, which are verified by Figure 15. This figure shows the average required CPU cycles per packet increase when the threshold becomes larger. What's more, when the threshold is less than 1000, Sketchtree requires the least CPU cycles per packet and possesses the fastest inserting speed. When the threshold is large, *i.e.*, 1500, our experimental results in Figure 12 show that very few flows are recorded by Sketch 0.

2) Sensitivity Analysis for Optimization 1: In this section, we observe the impacts of Optimization 1 on different metrics. That is, Filters 1 and 2 measure aggregate mice flows with attribute of destination address and source address, respectively. For ease of reading, Sketchtree combined with Optimization 1 is denoted by Sketchtree+Opt1 in the figures.

Impact on the number of covered flows on each sketch of Sketchtree (Figure 16): For Sketch 1, it covers 500K flows, which take up 70% of total flows. Compared with the results without Optimization 1 in Figure 8, the number of covered flows in Sketch 1 increases by 150K, which is more than 20% of total flows. On the contrary, for Sketch 2, the number of its covered flows decreases from 200K (results in Figure 8) to less than 50K flows. Most of its lost flows are measured by Sketch 1, which requires less computational overhead to process a packet. Thus, this optimization method enables nearly 150 K flows to be recorded faster.

Impact on the number of covered packets on each sketch of Sketchtree (Figure 17): the

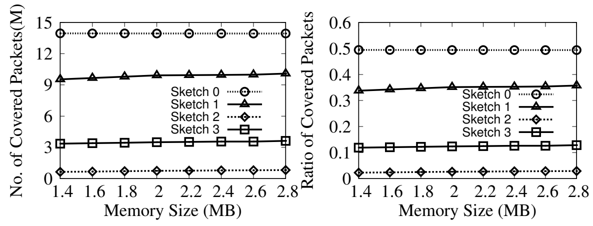


Fig. 17: Packet Distribution on Each Sketch of Sketchtree

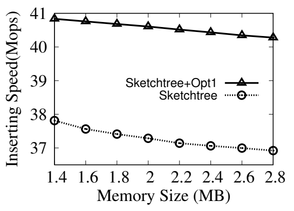


Fig. 18: Inserting Speed vs. Memory Size

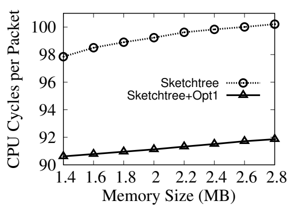


Fig. 19: CPU Cycles per Packet vs.Memory Size

results show that Sketch 1 covers more than 9M packets, which are nearly 40% of all the packets. Compared with the results without Optimization 1 in Figure 9, where Sketch 1 records 6M packets, our optimization method helps to cover 3M more packets and increase the ratio of covered packets by nearly 50%. On the contrary, the number of covered packets in Sketch 2 drops from 3M packets (without Optimization 1) to 0.8M packets. That means nearly 10% of total packets originally in Sketch 2 are now mainly processed by Sketch 1 after Optimization 1. Moreover, the number of covered packets in Sketch 3 is not sensitive to Optimization 1, with no obvious change compared with the results without Optimization 1 in Figure 9.

Impact on inserting speed and CPU cycles per packet (Figures 18-19): the results show that Optimization 1 can obviously improve the inserting speed and decrease the average required CPU cycles per packet. For example, Figure 4 shows that Optimization 1 increases inserting speed by 3 Mops on average. That's because the number of packets processed by Sketch 1 increases while that processed by Sketch 2 decreases after Optimization 1 (Sketch 1 is closer to the root node than Sketch 2 in the architecture of Sketchtree, so Sketchtree processes a packet measured by Sketch 1 faster than that measured by Sketch 2). In Figure 19, Optimization 1 can reduce the average required CPU cycles per packet by nearly 8, which takes up 8% of CPU cycles per packet without Optimization 1.

Summaries: (1) The value of threshold $\mathcal{T}$ determines the distribution of flows/packets on Sketches 0 to 3, and furthermore, influences the measurement accuracy of each sketch in Sketchtree. In particular, a larger threshold promises higher measurement accuracy of elephant flows. (2) The smaller the threshold is, the faster Sketchtree processes a packet. (3) Optimization 1 can help Sketchtree to decrease the computational overhead per packet by 10%.

7. Conclusion

In this paper, we propose an accurate traffic measurement framework named Sketchtree to record both elephant flows and aggregate mice flows with high accuracy. In particular, Sketchtree inherits the idea of filtering and develops it into the multi-attribute filtering. These filters are organized hierarchically, which can gradually reduce the number of flows through filters one by one and the probability of hash collision in traffic measurement. We also present some optimization methods to further improve the resource efficiency of Sketchtree. Experimental results show that Sketchtree can largely reduce measurement errors of aggregate mice flows, and maintain high measurement accuracy of elephant flows compared with state-of-the-art solutions under the same memory consumption.

References

[1] Gongming Zhao, Liusheng Huang, Zhuolong Yu, Hongli Xu, and Pengzhan Wang. On the effect of flow table size and controller capacity on sdn network throughput. In *ICC*, pages 1–6. IEEE, 2017.

[2] Span. http://www.cisco.com/c/en/us/tech/lan-switching/switched-portanalyzer-span/index.html.

[3] Renato V Iozzo and James D San Antonio. Heparan sulfate proteoglycans: heavy hitters in the angiogenesis arena. *The Journal of clinical investigation*, 108(3):349–355, 2001.

[4] S. Chen H. Xu, H. Huang and G. Zhao. Scalable software-defined networking through hybrid switching. In *INFOCOM, 2017 Proceedings IEEE*. IEEE, 2017.

[5] Sílvia Farraposoα, Philippe Owezarskiβ, and Edmundo Monteiroγ. An approach to detect traffic anomalies.

[6] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[7] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 217–228. ACM, 2005.

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

[9] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. Cold filter: A meta-framework for faster and more accurate stream processing. In *Proceedings of the 2018 International Conference on Management of Data*, pages 741–756. ACM, 2018.

[10] Jeffrey Scott Vitter. Design and analysis of dynamic huffman codes. *Journal of the ACM (JACM)*, 34(4):825–845, 1987.

[11] Smriti R Ramakrishnan, Garret Swart, and Aleksey Urmanov. Balancing reducer skew in mapreduce workloads using progressive sampling. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 16. ACM, 2012.

[12] Sudipto Guha and Boulos Harb. Wavelet synopsis for data streams: minimizing non-euclidean error. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 88–97. ACM, 2005.

[13] Anna C Gilbert, Yannis Kotidis, S Muthukrishnan, and Martin Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Vldb*, volume 1, pages 79–88, 2001.

[14] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 20–29. ACM, 1996.

[15] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[16] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.

[17] Qun Huang, Xin Jin, Patrick PC Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. Sketchvisor: Robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 113–126. ACM, 2017.

[18] Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1093–1103. Association for Computational Linguistics, 2012.

[19] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011.

[20] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, pages 398–412. Springer, 2005.

[21] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1449–1463. ACM, 2016.

[22] Graham Cormode. Sketch techniques for approximate query processing. *Foundations and Trends in Databases. NOW publishers*, 2011.

[23] Robert Schweller, Ashish Gupta, Elliot Parsons, and Yan Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 207–212. ACM, 2004.

[24] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Computing Surveys (CSUR)*, 39(1):3, 2007.

[25] Jihwan Kim, Ajay Balankutty, Amr Elshazly, Yan-Yu Huang, Hang Song, Kai Yu, and Frank O'Mahony. 3.5 a 16-to-40gb/s quarter-rate nrz/pam4 dual-mode transmitter in 14nm cmos. In *Solid-State Circuits Conference-(ISSCC), 2015 IEEE International*, pages 1–3. IEEE, 2015.

[26] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *Nsdi*, pages 311–324, 2016.

[27] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[28] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A Dinda, Ming-Yang Kao, and Gokhan Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Transactions on Networking (ToN)*, 15(5):1059–1072, 2007.

[29] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114. ACM, 2016.

[30] Nuno Homem and Joao Paulo Carvalho. Finding top-k elements in data streams. *Information Sciences*, 180(24):4958–4974, 2010.

[31] Graham Cormode and S Muthukrishnan. What's new: Finding significant differences in network data streams. *IEEE/ACM Transactions on Networking (TON)*, 13(6):1219–1232, 2005.

[32] Matthew V Mahoney. Network traffic anomaly detection based on packet bytes. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 346–350. ACM, 2003.

[33] Sudipto Guha and Andrew McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5):2044–2059, 2009.

[34] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. Persistent data sketching. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 795–810. ACM, 2015.

[35] Open vswitch: open virtual switch. http://openvswitch.org/.