Accurate Traffic Measurement with Hierarchical Filtering

SIGMETRICS 2019 Submission

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 prefix) 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 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, called **Sketchtree**, with multiple filters 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. Moreover, we also design some mechanisms to improve the resource efficiency. To validate our proposal, we have implemented Sketchtree with C, and conducted experiments using real campus traffic traces. The experimental results show that Sketchtree can improve the measurement accuracy of aggregate mice flows 2.1 times on average, and preserve high measurement accuracy of elephant flows and processing speed, compared with state-of-the-art sketches.

CCS Concepts

•**Networks** → **Network measurement;**

Keywords

Network Measurement; Hierarchical Filtering; Sketch; Attribute

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, many different measurement methods, *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 methods 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 [33]) 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 in high-speed networks.

On the contrary, sketches provide an alternative solution to achieve fine-grained traffic measurement with compact data structures, which can summarize traffic statistics of all packets with fixed-size memory. Due to resource constraints on switches, most current sketches focus on the statistics of elephant flows, as these flows are usually relevant in many network optimization tasks, such as heavy-hitters detection [18] and load balancing. Specifically, heavy hitter detection identifies the large flows whose byte volumes are above a threshold, while load balancing can be achieved by redirecting some elephant flows [15].

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 information of aggregate mice flows with different attributes (*e.g.*, all the mice flows with the same source IP prefix) [10], regardless of the information of each individual mice flow. For example, during DDoS attack, a number of sources send packets to a restricted destination address [24]. To detect this attack, we need the overall traffic amount of (mice) flows to a destination address. For convenience, we call these mice flows aggregated with the attribute of destination address. Another example is network scan [20], which is the opposite of DDoS. It requires the overall traffic of mice flows from a specific source address (*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 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 [34]) show that the average measurement error of mice flows is several times of their real frequency. Specifically, when the mice flow is misreported as an elephant flow, the estimation error may be larger, *e.g.*,

© 2018 ACM. ISBN 978-1-4503-2138-9.

DOI: 10.1145/1235

one to hundreds of times. Thus, the sum of all individual flows in an aggregate flow may violate its real frequency a lot. (2) Sometimes all the individual flows in an aggregate mice flow are difficult to be figured out. For example, to estimate an aggregate flow to a certain destination address 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 task. Specifically, elephant flows can be recorded by Count-Min. To detect DDoS attacks, an additional Count-Min sketch needs to be applied to record the 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 method needs each sketch to process all the packets in the data stream independently, which may be 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. Specifically, this framework comprises multiple filters, each associated with a specific task. For each task, the corresponding filter will preprocess flows and send the relevant flows 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 conduct separation for 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 measure both elephant flows and aggregate mice flows with high accuracy.

- To decrease the memory and computation consumption, we design some optimization methods, such as building Huffman tree [30] by adjusting the orders of attributes in Section 4.3.1 and separating hash values in Section 4.3.2. With these methods, our solution can be 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 and Cold Filter, under the same memory size.

2. BACKGROUND AND MOTIVATION

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

2.1 Related Works and Typical Sketches

Network measurement has been extensively studied in the context of different techniques such as sampling [26], wavelets [11,13], and sketches [3, 5, 8, 17, 32]. Among these techniques, sketch is widely used for its high processing speed, low memory consumption and high measurement accuracy for (elephant) flows. The most acknowledged and widely used sketch is Count-Min (CM) [8]. Specifically, Count-Min 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 [12], was designed. Different from Count-Min, CM-CU 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 [5], a hierarchical data structure [7], or an array with k counters [23, 27]. However, these sketches can achieve high precessing speed when the number of elephant flows is small, *e.g.*, 32 in ASketch [27]. 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 [32]. The survey of more sketches can be found in [6].

Instead of processing data stream in one sketch, recent research called Cold Filter [34] 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 steam processing algorithm, *e.g.*, CM-CU [12] and Space-Saving [23]. However, Cold Filter is mainly designed to improve the measurement accuracy of elephant flows, paying little attention to the measurement of aggregate mice flows.

2.2 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 attach great importance 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** [18]: identify elephant flows that consume more than a threshold of link capacity during a time window.

- **Traffic changes detection** [28]: indicate flows which trigger the most traffic changes over two consecutive time windows. This task requires the information of elephant flows.

- **DDoS** [25]: [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** [20]: 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/applications require accurate information of elephant flows or aggregate mice flows. For example, Heavy Hitters needs the traffic information of elephant flows whose byte volumes are above a predefined threshold, *e.g.*, 1% of the link capacity during a time window. Another example is network scan, which requires the traffic information of aggregate mice flows with the attribute of source address. That is, what's important is whether the overall traffic of mice flows from one source address exceeds the threshold, 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. Since aggregate flows comprise of individual flows, one may consider that current sketches can achieve adequate accuracy for individual flows, such that there is no need to measure the aggregate flows. However, our experiments on some typical sketches, *e.g.*, Count-Min, CM-CU and Cold Filter, using real campus traffic traces show that the relative measurement error for individual mice flows is in range of 200%-1000%, which is unacceptable for the listed tasks. Therefore, we need to measure both elephant flows and aggregate mice flows (with different attributes) accurately, which is the main contribution of this paper.

2.3 Hash Collision

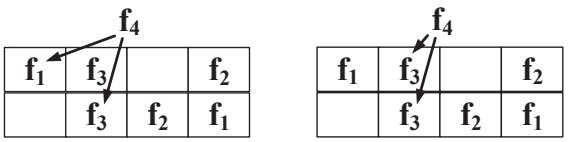
Typically, most sketches, *e.g.*, CM and CM-CU, use multiple hash functions to summarize massive data streams within a limited memory size. As a result, measurement error in these sketches is mainly caused by hash collision among different flows, which may lead to inaccurate measurement results. There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

- Error A: it will not mistake the elephant flow and mice flow, but give relatively inaccurate results for a flow.
- Error B: it may misclassify a mice flow and report it as an elephant flow.

Here, we give an example to illustrate four kinds of hash collisions in Figure 1, and analyze how these hash collisions cause different types of errors. In the example, there are two rows and two hash functions in the sketch, one for each row. Different packets/flows (flows are recorded packet by packet, and we do not distinguish these two definitions for simplicity) are hashed into different counters. We just show the flow in the counter and omit the detailed frequency.

As shown in Figure 1(a), for the incoming mice flow f_4, it is hashed into the counters which are shared by flows f_1 and f_3 in the first and second rows, respectively. The sketch will report the minimum value as the measurement result, *i.e.*, $\min\{N_{f_1}+N_{f_4}, N_{f_3}+N_{f_4}\}$.

[1] Due to the diversity of attacking methods, *e.g.*, ICMP, LAND, IP, *etc.* [10], 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 [10, 20].



(a) hash collision with different flows (b) hash collision with one flow

Figure 1: Two examples of hash collisions

- Case 1: if at least one of f_1 and f_3 is a mice flow, the measurement result, *i.e.*, $\min\{N_{f_1}+N_{f_4}, N_{f_3}+N_{f_4}\}$, will not violate the real frequency of flow f_4 a lot. Thus, it belongs to Error A.
- Case 2: if both f_1 and f_3 are elephant flows, the estimation result $\min\{N_{f_1}+N_{f_4}, N_{f_3}+N_{f_4}\}$ is much larger than the real frequency of flow f_4, *i.e.*, N_{f_4}. As a result, flow f_4 will be mistakenly regarded as an elephant flow, which belongs to Error B.

Figure 1(b) shows that the incoming mice flow f_4 collides with one flow f_3 on both two rows. Since this hash collision is a special case of Figure 1(a), it can also be divided into two cases and two corresponding errors like Figure 1(a).

The error of Case 2 belongs to Error B, which is more serious than Error A as the traffic size of an elephant flows 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. For Case 1, the estimation error 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.

2.4 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 [34]. 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 or even higher speed [19], 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 [21] can detect heavy changes by comparing the frequency of a flow in two adjacent time windows relying on an Invertible Bloom Lookup Table (LBLT) [21]. However, its computational overhead and throughput are 2584 CPU cycles per packet and 2.5Gbps [17], respectively. What's more, RevSketch [29]

sketch	CPU cycle	sketch	CPU cycle
Bloom Filter [4]	77	CountSketch [5]	175
Count-Min	78	FlowRadar [21]	2584
CM-CU	97	RevSketch [29]	3858
Cold Filter	102	Unimon [22]	4382
FSS [16]	120	Deltoid [9]	10454

Table 1: Comparison on computational overhead of existing sketches

can infer the IDs of top-k flows. But, it requires 3858 CPU cycles for processing a packet, and its throughput is only 4.90Gbps [17]. Some computational overheads of existing sketches are shown in Table 1[2]. (2) More seriously, multiple sketches may be deployed on the same 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 through 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), and is usually limited.

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 needs the information of 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, namely, LGM flows, because only these LGM flows can be potential attacks/dangers. Among all the t 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. Specifically, 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.

[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.

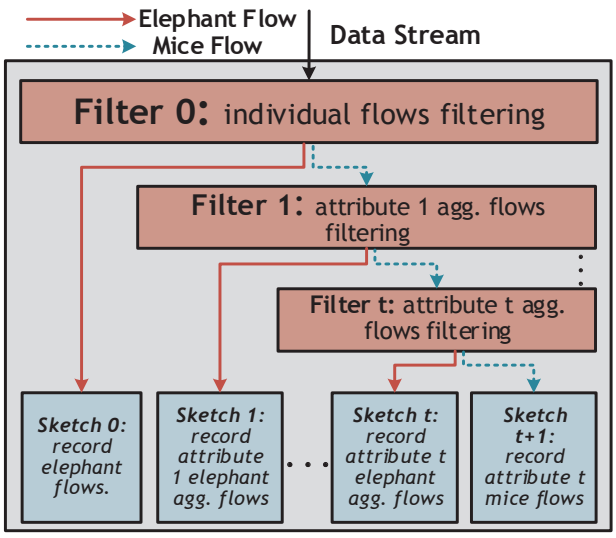


Figure 2: Sketchtree Architecture

Figure 2 presents the architecture of Sketchtree. Specifically, Sketchtree consists of two stages: filtering stage and measurement stage. Generally, filtering stage includes $t+1$ filters, namely, Filter 0, Filter 1,..., Filter t, and measurement stage consists of $t+2$ sketches, namely, Sketch 0, Sketch 1,..., Sketch $t+1$. Each filter is responsible for flow separation and each sketch is responsible for flow measurement. Moreover, Filter i ($0 \leq i \leq t-1$) will send the LGM flows (or elephant flows when $i = 0$) to the associated sketch (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 measuring the information of individual mice flows. 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, the cost for the measurement of these flows is not too much. 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 about 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 the 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 distinguish 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. Here are the explanations.

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. Combined with the cases of hash collision in Section 2.3, this mutual-independent measurement method guarantees that hash collisions happen with a very small probability. For example, most of the flows are swapped out by Filter 0 and only the estimated elephant flows are sent to Sketch 0. Since the number of flows has been drastically reduced in Sketch 0, an elephant flow is not likely to collide with other flows. Thus, the measurement accuracy can be guaranteed.

Resource efficiency: there are multiple sketches and filters in Sketchtree. One may think that the memory and computation consumption is much more than individual sketches or filters. However, we argue that filter, itself, is an efficient measurement method and filter-based measurement can, in fact, reduce the resource usage, compared with a single sketch under the same accuracy requirement. What's more, we present some optimization methods to further reduce the resource usage in Sections 4.3.1 and 4.3.2.

Adaptivity protect: there are t attributes, which may be diverse under different tasks. That is, these attributes are task-oriented. Moreover, 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 and some circumstances where tasks are with specific priorities.

4. OUR SOLUTION

4.1 Algorithm

We give necessary explanations about some parameters.

- 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 value of flow f in Filter k, with $0 \leq k \leq t$. For example, $value_0(f)$ means the estimated value of flow f in Filter 0.
- $\mathcal{F}_{k,f}$: the aggregate flow that flow f belongs to under attribute 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 value 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. Filter 1 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.

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) > T_0 then
4:    Send the flow to Sketch 0
5: else
6:    Update process for mice/aggregate flows
7:    k = 1 /*k is the attribute index*/
8:    while k ≤ t do
9:       if value_k(F_{k,f}) > T_k then
10:         Send the flow to Sketch k
11:         Sketch k records the value of aggregate flow 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 value of flow f
17:      k = k + 1
18:      Aggregate flow F_{k,f} is recorded by Filter k
```

4.2 Filter Design and Sketch Selection

Strawman Solution for Filter Design: one natural solution is to use a common sketch, *e.g.*, CM, 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 [34]. (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 [34], 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, the filter works as follows:

- 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 packet/flow is $value_1 + 1$.
- If $v_1 \geq \mathcal{T}'$, the filter records the packet 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 packet/flow overflows and will be regarded as an elephant flow.

Sketch Selection: Each sketch records the filtered flows, including elephant flows and large GM 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 presented in Section 6.2.

4.3 Two Optimization Methods for Resource Efficiency

Sketchtree consumes memory and computational resources, which may be scarce on switches. Thus, we propose two optimization methods to make Sketchtree work more efficiently.

4.3.1 Optimization 1: Organizing Sketchtree as a Huffman Tree

Sketchtree can be recognized as a binary tree, where Filter 0 is the root of the tree. In particular, each filter is the internal node and each sketch is the leaf node. Take Filter 1 as an example. If the estimated value 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, the right child node, *i.e.*, filter 2. Therefore, the processing pipeline can be regarded as the search operation in the binary tree. As we know, the computing ability is a scarce resource on switches. To reduce the processing computation for data stream, we hope that the expected computation for 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 computation overhead.

Let's consider the placement of sketches 1 to t. Each parent node (filter) records the aggregate flows with a distinct attribute, which associates with a respective task. According to the architecture of Sketchtree, the aggregate rule can be placed in different orders. Thus, we can exchange the placement of each parent node in order to minimize the total computation overhead. 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 computation overhead. Moreover, the adjustment can maintain the stableness of Sketchtree as we only need to exchange the index of the filter and sketch so as to improve throughput of Sketchtree.

4.3.2 Optimization 2: Saving Memory and Computation Usage

Memory-saving in the filtering stage: each filter needs to distinguish elephant flows from the data stream rather than 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 counter stops counting. Thus the size of a counter can be δ, with $2^\delta - 1 = \mathcal{T}$. For example, for the first layer in our design filter, $\mathcal{T} = 15$ and $\delta = 4$. Under this circumstance, we can 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 [34]. As a result, $\frac{7}{8}$ of the memory size for Layer 1 is saved.

Memory-saving in the measurement stage: after filtering, the frequencies of most of (aggregate) flows will not exceed the threshold, and only a minority of (aggregate) flows will be sent to the respective 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 in sketch 0 will be much less than the total number of flows in the dataset. Since the number of filtered flows is much reduced, most of the counters remain unused and we can reduce the number of rows d to save memory usage. In particular, the original approach is to locate one counter in each row with totally d hash operations. Our approach is to locate d counters in one row, with totally d hash operations as well. As a result, our approach can save $d-1$ rows of counters under the same number of hash operations.

Computation-saving: relevant literatures and our experiments show that most computational overhead lies in hash operations. Thus we expect to reduce hash computations and our optimization method can locate h counters by one hash function. Specifically, we split the large-bit hash value (usually 32 bits [34]) 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. ANALYSIS OF SKETCHTREE

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:

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

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

Standard Bloom filter: A standard Bloom filter [4] 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 [4]:

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

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 in this time window, note that $J_k[j]$ is the subset of flows whose current estimated frequencies are not less than k. Formally, $J_k[j] = \{f | \hat{N}_f[j] \ge k, k \in Z^+\}$.*

LEMMA 1. *The subsets of flows, $I_k[j]$ and $J_k[j]$, have the following relation:*

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

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 $|I_k[j]| \le |J_k[j]|$ 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 $\sum_{i=0}^{x-1} P_{f_p}(w,d,i) = \bar{P}_{f_p}(w,d,x) \cdot x$.

Due to the false positive rate, the estimated frequency of a flow may be larger than its real frequency. Similar to [34], 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. □

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

LEMMA 2. *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.*

PROOF. Similar to [34], $P_{f_p}(w,d,n)$ 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. □

Since $\bar{P}_{f_p}(w,d,x)$ is a monotonic function of x, $|J_k[j]|$ can be bounded recursively by Lemma 2. For simplicity, note that $|J_k[j]|^L$ and $|J_k[j]|^U$ are the lower bound and upper bound 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:*

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

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:

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

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:

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

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:

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

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 [14, 31] 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 :

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

According to Lemma 1, $|J_i[t_u]|$ is bounded by $|I_i[t_u]|$. Thus the misreport rate P_{mr} is bounded under the random order model.

6. PERFORMANCE EVALUATION

6.1 Experimental Setup

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

Measurement Tasks: despite the measurement of individual flows for tasks like flow size distribution and entropy estimation, we also adopt two security-associated tasks, *i.e.*, DDoS and network scan, as the source and destination addresses are common in host based attacks. Thus, Sketchtree contains two attributes of aggregation, *i.e.*, *t*=2. According to Optimization 1 in Section 4.3.1, the order of attributes can be adjusted to minimize the computational overhead of Sketchtree. At this section, we put the source address as the first attribute to observe the performance of Sketchtree without Optimization 1. Then we will estimate the effect of Optimization 1 in Section 6.5.

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

Table 2: Parameter setting for Sketchtree

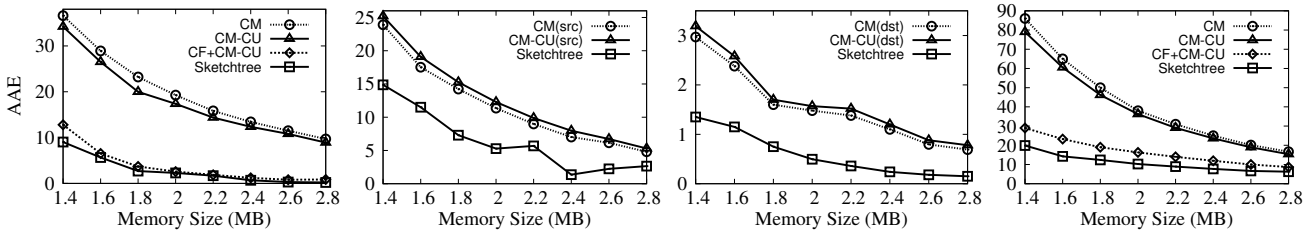
Implementation: we have implemented CM (Count-Min), CM-CU, CF+CM-CU (Cold Filter+CM-CU) and our Sketchtree with C. In our implementation, we find that the ability to process packets for Open vSwitch (OVS) [1] is constrained by software interrupts of the NIC driver, and the inserting speed is relatively trivial. Therefore, we build individual sketch implements and directly feed them data stream, despite that we have successfully embedded the C sketch module into OVS. Also, the real traffic traces cannot be built by testbed as the source address and destination address are numerous. Thus, we execute the C procedure by directly reading the traffic file.

6.2 Benchmark Details and Parameter Settings

Sketchtree: Sketchtree mainly focuses on the measurement of elephant flows and two kinds of aggregate mice flows. For ease of expression, we say that Sketch 0, 1, and 2 record **elephant flows, LBG1 flows and LBG2 flows**, respectively. As a by-product, we also give the measurement accuracy of individual mice flows in Filter 0 and Sketch 4, these flows are called **mice flows**. 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 [34]. 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, $\mathcal{T}$ is the threshold in a layer, M is the memory size of the layer and h is the number of hash functions in each layer. Table 2 lists the parameter settings for each sketch/filter.

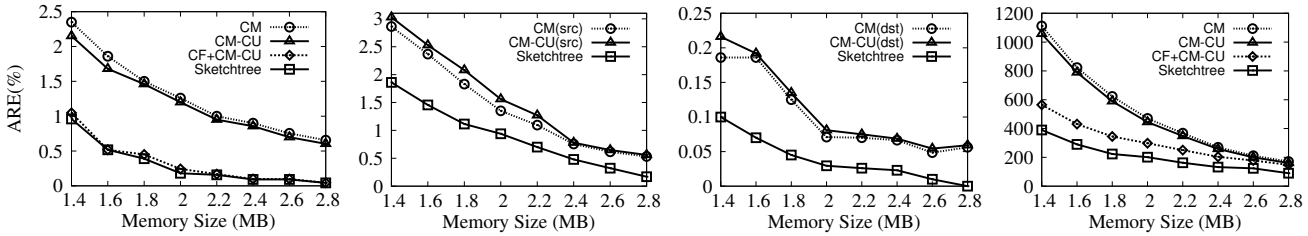
CM/CM-CU: both two sketches are used to measure individual flows, including elephant flows and mice flows. CM/CM-CU are allocated 5 hash functions to explore as high accuracy of elephant/mice flows as possible. The size of each counter is 32 bits [34]. The value of columns w depends on the memory size, which ranges from 1.4 MB to 2.8 MB.

CF+CM-CU: CF+CM-CU represents the framework in which CF (Cold Filter [34]) 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. It is composed of a CF for mice-flow measurement (also for filtering) and CM-CU for recording elephant flows. Let M_{cf} be the memory size of CF, and M_t be the memory size of CM-CU. As recommended in [34], $M_{cf} : M_t = 9 : 10$. Specifically, CF includes



(a) Elephant Flows (b) First Aggregate Flows (c) Second Aggregate Flows (d) Mice Flows

Figure 3: AAE vs. Memory Size for Four Sketches



(a) Elephant Flows (b) First Aggregate Flows (c) Second Aggregate Flows (d) Mice Flows

Figure 4: ARE vs. Memory Size for Four Sketches

two layers with parameter setting the same as filter 1 in Sketchtree. The only difference is that CF is set with threshold $\mathcal{T} = 256$ in the second layer [34]. The CM-CU contains 3 hash functions, each with an array of counters. The size of each counter is 32 bits. Since the optimization method named aggregate-and-report in CF can be applied to all these sketches to handle traffic bursts, all these sketches are compared without aggregate-and-report for fairness.

CM(src/dst): Sketch 1 and Sketch 2 in Sketchtree need to record the frequency of aggregate mice flows. For comparison, we use CM to estimate frequency of aggregate mice flows with two attributes, *i.e.*, source and destination address, namely CM(src) and CM(dst), respectively. Both CM(src) and CM(dst) are equipped with 3 hash functions with parameter settings the same as those of CM. For fairness, the total memory size of CM(src) and CM(dst) are equal to that of other sketches.

CM-CU(src/dst): CM-CU(src) and CM-CU(dst) are adopted to record the frequency of aggregate mice flows with two attributes, *i.e.*, source and destination address, respectively. Moreover, the parameter settings of CM-CU(src/dst) are the same as those of CM(src/dst).

6.3 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 Estimation: 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: 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 covered flows to the dataset. Moreover, we use the number of covered packets in each sketch to shown the processing load for each sketch, and also to show the distribution of traffic in each sketch.

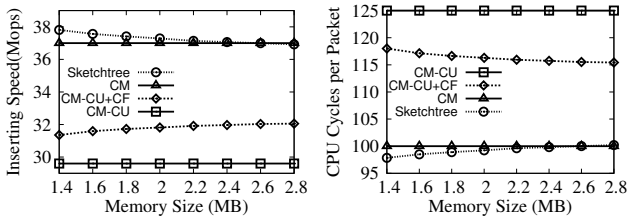


Figure 5: Inserting Speed vs. Memory Size

Figure 6: Memory Size vs. CPU Cycles per Packet

6.4 Evaluation Results

Inserting Speed and CPU Cycles per Packet (Figures 5-6): we estimate the computational overhead for each sketch, and the results are shown in Figures 5 and 6. As we can see, Sketchtree possesses fastest inserting speed, compared with other three benchmarks. Specifically, the inserting speeds of Sketchtree and CM both 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 advantages 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 3(a)-3(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 in Sketch 0, Sketchtree achieves the lowest AAE, which is only 80%, 11% and 13% of AAE in CF+CM-CU, CM and CM-CU, when the memory size is 2 MB; (2) for the measurement of the first aggregate flows in Sketch 1, Sketchtree reduces AAE by approximately 45% and 40% compared

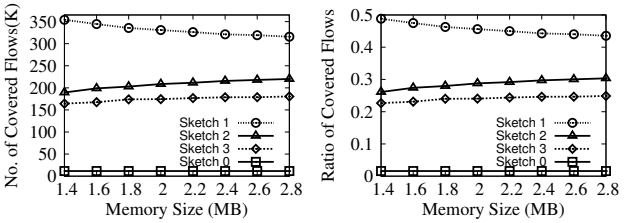


Figure 8: Memory Size vs. Flow Distribution in Each Sketch

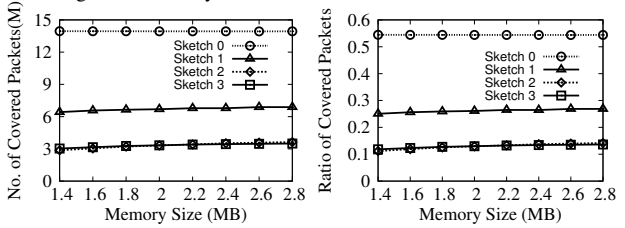


Figure 9: Memory Size vs. Packet Distribution in Each Sketch

with CM(src) and CM-CU(src) when the memory size ranges from 1.4 MB to 2.8 MB; (3) for the measurement accuracy of the second aggregate flows in Sketch 2, Sketchtree dramatically outperforms CM(dst) and CM-CU(dst), reducing AAE by nearly 60% compared with these two sketches.

ARE for Elephant Flows and Aggregate Mice Flows (Figures 4(a)-4(c)): we observe the influence of memory size on ARE by increasing the memory size of each benchmark from 1.4 MB to 2.8 MB. The results are shown in Figure 4. As we can see, ARE of each sketch decreases as the memory size increases. That's because a larger memory size contains more counters and can avoid hash collisions with larger probability. For ARE in Sketch 0 (Figure 4(a)), Sketchtree achieves close performance to CF+CM-CU, which is a state-of-the-art sketch for elephant-flow measurement. By comparison, CM and CM-CU cannot record elephant flows with high accuracy, increasing ARE by about 150% compared with CF+CM-CU and Sketchtree. For the measurement accuracy of aggregate mice flows in Sketch 1 (Figure 4(c)), Sketchtree outperforms CM(src) and CM-CU(src), reducing ARE by averagely 50% and 46%, respectively. Moreover, Sketchtree reduces ARE of aggregate mice flows in Sketch 2 by averagely 70% and 67% (Figure 4(d)), compared with CM(dst) and CM-CU(dst), respectively.

AAE and ARE for Mice Flow Measurement (Figures 3(d) and 4(d)): As a by-product, we consider the performance of Sketchtree in terms of the measurement accuracy of individual mice flows. As shown in Figure 3(d), CM and CM-CU possess very large AAE, for example, 39 and 38 respectively when the memory size is 2 MB. While CF+CM-CU achieves only half of AAE compared with CM. What's more, AAE of Sketchtree is 1.5 times and 4 times lower than CF+CM-CU and CM, respectively. In particular, when the memory size exceeds 2 MB, only AAE of Sketchtree is less than 10. As for ARE in Figure 4(d), our solution, Sketchtree, performs best, and ARE of Sketchtree is 88% when the memory size is 2.8 MB, which is very small as the number of mice flows are too big to record and the frequency of mice flows is low. On the contrary, CF+CM-CU, CM and CM-CU have trouble recording mice flows precisely, increasing ARE by averagely 450%, 400% and 52% compared with Sketchtree, respectively. 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, namely, Sketch 3.

Number of Covered Flows in Each Sketch (Figure 8): the dataset includes total 720K distinct flows. These flows are processed by four sketches in Figure 8. As we can see, nearly 50% of flows are processed by Sketch 1. These flows are aggregated by their source IP and these aggregate flows are recorded in Sketch 1. Moreover, Sketch 2 and Sketch 3 precess 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 in Each Sketch (Figure 9): Our campus dataset contains 28M packets, these packets are mainly processed by Sketch 0 by Figure 9. Combined with the results in Figure 8, Sketch 0 only records 1% of all flows, but its processed packets take 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. According to the flow distribution in Figure 8, although Sketch 2 and Sketch 3 individually record over 22% of flows, their precessed packets are relatively low, which can largely reduce the computational overhead for Sketchtree, for the number of packets in the bottom sketch is crucial to the overall computation of Sketchtree.

Summaries: (1) Sketchtree achieves the highest measurement accuracy for aggregate flows, reducing measurement error (AAE and ARE) by nearly 50% compared with existing sketches (CM(src/dst) and CM-CU(src/dst)). (2) Sketchtree can also record elephant flows accurately, reducing AAE (or ARE) by 10%-20% compared with CF+CM-CU, which is the state-of-the-art sketch that excels at 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 requires the least CPU cycles for recording a packet among all benchmarks. The reason is that most of the packets are processed by the first two sketches, *i.e.*, Sketches 0 and 1, which consume less computational overhead compared with the last two sketches, *i.e.*, Sketches 2 and 3.

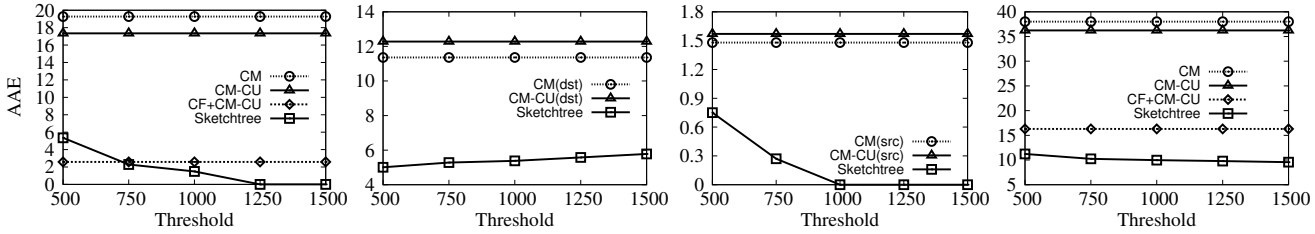
6.5 Sensitivity Analysis

In this section, we give some experimental results to illustrate the impacts of parameter settings and Optimization 1 in Section 4.3.1.

6.5.1 Impacts of Threshold $\mathcal{T}$

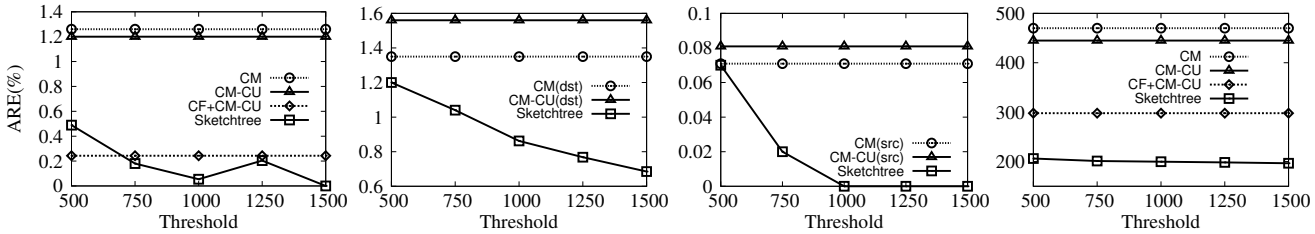
Impact on AAE (Figure 7): we conduct experiments by changing the threshold from 500 to 1500. Since other benchmarks are not affected by the threshold, AAE of these benchmarks is fixed. The results are shown in Figure 7. Generally, AAE in Sketch 0, Sketch 2 and Sketch 3 decreases while AEE in Sketch 1 increases, as the threshold increases. In particular, (1) in Sketch 0, when threshold exceeds 750, Sketchtree becomes the most accurate sketch. The reason is that the increased threshold makes less flow be regarded as elephant flows and furthermore reduces the probability of hash collisions in Sketch 0; (2) in Sketch 2, AAE drops quickly when the threshold increases from 500 to 1000; (3) In Sketch 1 and Sketch 3, AAE of Sketchtree increases and decreases slowly as the threshold increases, respectively. For example, in Sketch 1, AAE increases by 1.8 when threshold increases from 500 to 1500.

Impact on ARE (Figure 10): generally, Sketchtree outperforms CM, CM-CU and CF+CM-CU in terms of ARE, and ARE of Sketchtree decreases as the threshold increases for Sketch i, with $i = 0, 1, 2, 3$. In particular, (1) in Sketch 0, when the threshold increases to 750, the ARE of Sketchtree is lower than 0.2%, which is the least among all sketches; (2) in Sketch 1, as the threshold increases, ARE of Sketchtree decreases. However, AAE of Sketchtree in Sketch 1 increases by Figure 7(b). That's because the



(a) Elephant Flows (b) First Aggregate Flows (c) Second Aggregate Flows (d) Mice Flows

Figure 7: AAE vs. Threshold $\mathcal{T}$ for Four Sketches



(a) Elephant Flows (b) First Aggregate Flows (c) Second Aggregate Flows (d) Mice Flows

Figure 10: ARE vs. Threshold $\mathcal{T}$ for Four Sketches

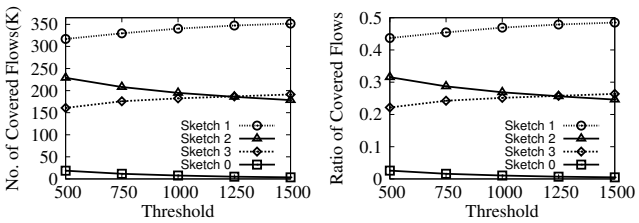


Figure 11: Threshold $\mathcal{T}$ vs. Flow Distribution in Each Sketch

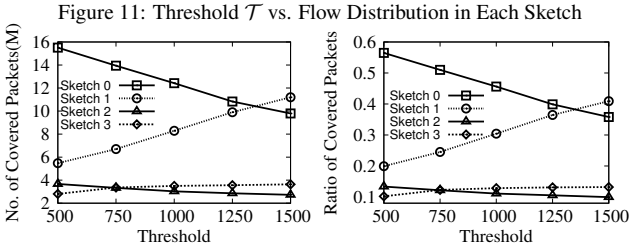


Figure 12: Threshold $\mathcal{T}$ vs. Packet Distribution in Each Sketch

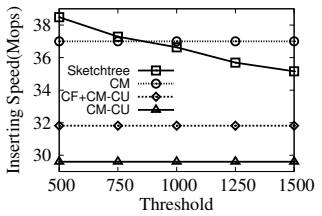


Figure 13: Threshold $\mathcal{T}$ vs. Inserting Speed

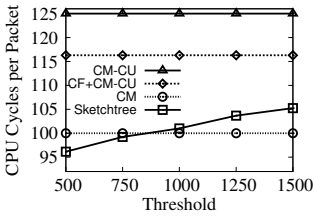


Figure 14: Threshold $\mathcal{T}$ vs. CPU Cycles per Packet

average frequency of elephant flows becomes larger, and therefore ARE becomes smaller. By comparison, (3) in Sketch 3, ARE decreases as the threshold increases, which is the same trend as the AAE in Sketch 3. The reason is that the number of mice flows is numerous and the average frequency of mice flows is not drastically affected by the threshold. Thus both AAE and ARE in Sketch 3 decrease as the threshold increases.

Impact on Number of Covered Flows (Figure 11): generally, our results show 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 losses 1.5K flows as the threshold increases from 500 to 1500, it losses nearly 80% of its covered flows in Figure 11. As for the ratio of covered flows (right plot in Figure 11), 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 Number of Covered Packets (Figure 12): generally, 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 flows increases, the absolute frequency of covered flows in Sketch 3 is not significant.

Impact on Inserting Speed and CPU Cycles (Figures 13-14): as shown in Figure 13, a larger threshold can decrease 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 is decreased, which are verified in Figure 14. This figure shows the average required CPU cycles per packet increase when the threshold becomes larger. What's more, if the threshold is less than 1000, Sketchtree requires the least CPU cycles for record-

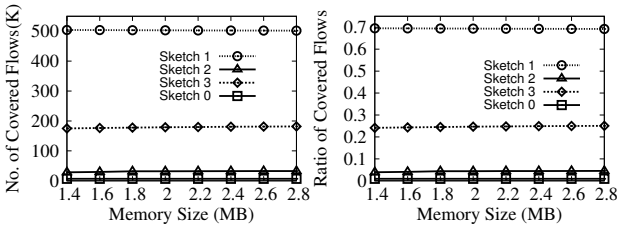


Figure 15: Flow Distribution in Each sketch

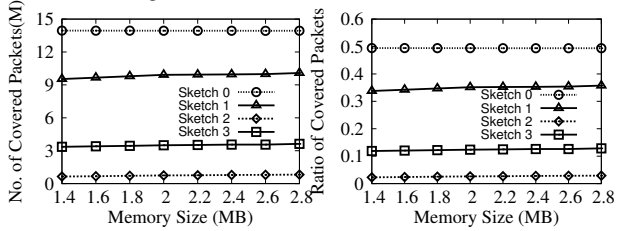


Figure 16: Packet Distribution in Each Sketch

ing a packet and possesses the fastest inserting speed. When the threshold is too large, *i.e.*, 1500, our experiment results in Figure 11 show that very few flows are recorded by Sketch 0. Thus, to explore the high inserting speed and record elephant flows in Sketch 0, a suitable threshold is required, *e.g.*, 750.

6.5.2 *Impacts of Optimization 1*

In this section, we observe the influence of Optimization 1 by exchanging the order of two attributes. That is, the first attribute is destination address and the second attribute is source address. For ease of reading, Sketchtree combined with optimization 1 is denoted by Sketchtree +Opt1 in the figures.

Impact on Number of Covered Flows (Figure 15): generally, Sketch 1 covers the majority of flows and Sketch 2 losses the most number of flows. In particular, (1) for Sketch 0, since the optimization method will not affect the first filter, the number of flows in Sketch 0 will be fixed; (2) 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, (3) in Sketch 2, the number of its covered flows decreases from 200K (results in Figure 8) to less than 50K flows. Moreover, most of its lost flows are measured by Sketch 1 after optimization 1. Since Sketch 1 requires less computational overhead to process a packet, this optimization method enables nearly 150 K flows to be recorded faster.

Impact on Number of Covered Packets (Figure 16): the results show that Sketch 1 covers more than 9M packets, which are nearly 40% of total packets. Compared with the results without optimization 1 in Figure 9, where Sketch 1 records 6M packets, our optimization method covers 3M more packets and increases the ratio of covered packets by nearly 50%. On the contrary, the number of covered packets in Sketch 2 drops from 3M packets (result 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. What's more, the number of covered packets in Sketch 3 is not sensitive to optimization 1, with no obvious change compared with the results in Figure 9 (result without optimization 1).

Impact on Inserting Speed and CPU Cycles per Packet (Figures 17 - 18): the results show that optimization 1 can obviously improve the inserting speed and decrease the average required CPU cycles for recording a packet. Moreover, optimization 1 averagely

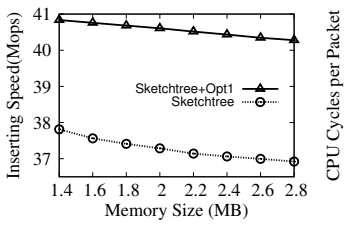


Figure 17: Memory Size vs. Inserting Speed

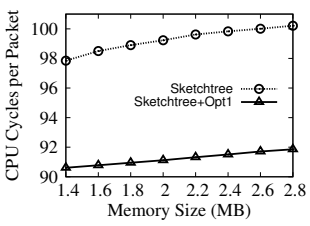


Figure 18: Memory Size vs. CPU Cycles per Packet

increases inserting speed by 3 Mops. That's because more packets are processed by Sketch 1, instead of Sketch 2 before optimization 1 (Figure 5). In Figure 18, optimization 1 can reduce the average required CPU cycles for processing a packet by nearly 8, which takes up 8% of CPU cycles for processing a 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) Sketchtree +Optimization 1 can decrease the computational overhead for recording a packet by 10%, compared with Sketchtree without Optimization 1.

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 to achieve high measurement accuracy for both elephant flows and aggregate mice flows. 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.

8. REFERENCES

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

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

[3] N. Alon, Y. Matias, and M. 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.

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

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

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

[7] G. Cormode, M. Garofalakis, P. J. Haas, C. Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011.

[8] 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.

[9] G. 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.

[10] S. Farraposoα, P. Owezarskiβ, and E. Monteiroγ. An approach to detect traffic anomalies.

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

[12] A. Goyal, H. Daumé III, and G. 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.

[13] S. Guha and B. 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.

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

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

[16] N. Homem and J. P. Carvalho. Finding top-k elements in data streams. *Information Sciences*, 180(24):4958–4974, 2010.

[17] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang. Sketchvisor: Robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 113–126. ACM, 2017.

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

[19] J. Kim, A. Balankutty, A. Elshazly, Y.-Y. Huang, H. Song, K. Yu, and F. 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.

[20] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 217–228. ACM, 2005.

[21] Y. Li, R. Miao, C. Kim, and M. Yu. Flowradar: A better netflow for data centers. In *Nsdi*, pages 311–324, 2016.

[22] 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. ACM, 2016.

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

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

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

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

[27] P. Roy, A. Khan, and G. 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.

[28] R. Schweller, A. Gupta, E. Parsons, and Y. 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.

[29] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. A. Dinda, M.-Y. Kao, and G. Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Transactions on Networking (ToN)*, 15(5):1059–1072, 2007.

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

[31] Z. Wei, G. Luo, K. Yi, X. Du, and J.-R. Wen. Persistent data sketching. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 795–810. ACM, 2015.

[32] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *NSDI*, volume 13, pages 29–42, 2013.

[33] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang. On the effect of flow table size and controller capacity on sdn network throughput. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–6. IEEE, 2017.

[34] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. 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.