Computer Communications 194 (2022) 148–155

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Tree sketch: An accurate and memory-efficient sketch for network-wide measurement

Lei Liu *, Tong Ding, Hui Feng, Zhongmin Yan, Xudong Lu

School of Software, Shandong University, Shunhua Road 1500, Jinan, 250101, China

ARTICLE INFO

Keywords:
Network measurement
Sketch
Memory-efficient
Quality of service

ABSTRACT

Traffic measurement provides essential information for bandwidth management and Quality of Service (QoS), which is an enabler of network status discovery. However, with the explosive growth of network traffic, traditional network measurement methods face challenges in terms of memory, computation, accuracy, especially in the high-speed networks. Network devices have no enough CPU or memory resources for collecting, counting and analyzing data streams, which makes measurement results inaccurate and unreliable. To address the problem, we design an accurate and memory-efficient Tree sketch. It consists of three different structures for heavy flow detection, Distributed Denial-of-Service (DDoS) attack detection, super-spreaders detection, respectively. The proposed heavy cuckoo algorithm stores the flows in categories according to their flow sizes reducing hash collisions in the sketch. Besides, Tree sketch employs the Single Instruction Multiple Data (SIMD) instructions to speed up packet processing. Experimental results show that F1 score of Tree sketch is 0.997 with 30 KB memory usage in heavy hitter detection, which is 0.019–0.137 higher than the state-of-the-art sketch algorithms. It is an efficient tool of traffic measurement in the backbone network and provides a good trade-off among accuracy, speed and memory usage.

1. Introduction

Traffic measurement plays a crucial role in maintaining network stability and guaranteeing the Quality of Service (QoS) [1,2]. In a computer network, flow-based monitoring not only detects elephant flows to improve routing, load balancing, and scheduling but also discovers potential threats (e.g., superspreaders [3–5] and Distributed Denial-of-Service (DDoS) attacks [6,7]) by monitoring fluctuations of network traffic. Measurement methods are mainly divided into two categories: sampling algorithms and data stream algorithms. Sampling algorithms select representative subsets from the original packets to estimate traffic statistics. Data stream algorithms count all packets using as few computing and memory resources as possible. Nowadays, these two methods and their variants are widely used in various measurement techniques [8–10].

Existing measurement approaches face three challenges as traffic speed and volume grow tremendously [11,12]. Firstly, network devices have only limited memory and cannot accommodate massive flows to count, so the measurement performance suffers severe degradation. Secondly, network hardware is constantly upgrading, and the packet processing speed does not keep pace with the ever-increasing line rate. Thirdly, with the popularization of distributed computing, single node measurement cannot accurately describe network status wide detection [13–15]. In recent years, many researchers have studied sketches to overcome these challenges. Sketch-based algorithms are typical data stream methods that use fixed-size memory to process per packet. The sketch is a compact data structure and consists of several independent hash tables. It maps each packet to the corresponding bucket through simple hash operations and performs addition operations in each bucket to get flow size. The sketch needs to count all packets in this process, so the measurement results are more accurate than the sampling algorithms. Besides, as long as the algorithm sets the same parameters for each sketch, it is easy to implement multiple-node network measurement by merging data in the sketch. The sketch structure provides a theoretical tradeoff between memory usage and accuracy.

Lots of efforts have been made to optimize the sketch algorithms to improve the performance of network measurement [16–19]. Cormode and S. Muthukrishnan [20] propose a CM sketch for summarizing data streams. The CM sketch reduces hash collisions by taking the minimum of multiple counters as the flow size. Unfortunately, when an elephant flow and a mouse flow are mapped into the same bucket, the mouse flow is greatly overestimated. Many algorithms are designed to decrease errors by separating the elephant flows from mouse flows [21–24]. Tang et al. [25] present the majority vote algorithm to filter

* Corresponding author.
E-mail address: l.liu@sdu.edu.cn (L. Liu).

https://doi.org/10.1016/j.comcom.2022.07.009
Received 14 February 2022; Received in revised form 1 June 2022; Accepted 6 July 2022
Available online 16 July 2022

mouse flows. Gong et al. [26] use a novel strategy called count-with-exponential-decay to discover elephant flows. Huang et al. [27] introduce SketchLearn, which finds elephant flows by automated statistical inference. If there is no elephant flow, mouse flows follow a Gaussian distribution in SketchLearn. Existing works have made great contributions, but there is still room for improvement. Sketch only stores one flow ID in each bucket. When multiple elephant flows are mapped to the same bucket, these algorithms only record one of the elephant flows and miss other flows. With limited memory resources, this way will cause a large number of false negatives.

To solve the problem mentioned above, we designed a Tree sketch, an accurate and memory-efficient measurement framework. The framework contains a novel algorithm called heavy cuckoo. Similar to cuckoo hashing [28], the algorithm uses two hash functions to diminish hash collisions. It hashes each packet to two buckets through two independent hash functions. If there are hash collisions in two buckets, the heavy cuckoo algorithm will discard the flow in the smaller bucket to another sketch. Tree sketch is divided into three components and allocates appropriate memory for each component to improve memory efficiency. It can perform multiple detection tasks to ensure the generality of the measurement framework and execute hash functions in parallel to accelerate the processing speed.

This paper makes the following contributions:

- We present a Tree sketch approach, which can achieve accurate network-wide measurement with limited resources. It implements four different measurement tasks simultaneously. Additionally, Tree sketch uses Single Instruction Multiple Data (SIMD) instructions to accelerate packet processing.
- We design a memory-efficient algorithm named Heavy Cuckoo. It divides all flows into three classes and stores them with appropriate memory. Even if the available memory is very small, the algorithm can find the elephant flows accurately in massive network traffic.
- We evaluate the performance of Tree sketch on real network traces. Experimental results show that Tree sketch has higher accuracy and smaller memory usage. Besides, it achieves accurate network-wide measurement.

The rest of the paper is organized as follows. Section 2 describes some basic concepts and related algorithms. Section 3 introduces the detailed design of the Tree sketch. Section 4 describes experimental results and provides an extensive evaluation of the performance of the Tree sketch. In the end, Section 5 concludes the paper.

2. Preliminaries

This section introduces fundamental works, including measurement tasks, the sketch, and algorithms for finding frequent elements. We can monitor the network in real-time and identify specific flows by performing the following common tasks. The sketch is the basic structure of the research work in this paper, and it can achieve efficient and accurate measurement by occupying a small amount of memory. Finding frequent items in the network is the key to congestion control and anomaly detection.

2.1. Measurement tasks

Some measurement tools can only support a specific measurement task, but these methods no longer meet the requirements of hardware vendors. If they want to extend other functions on these tools, vendors will have to pay a high price. So new measurement frameworks should be generic. Hence, one data structure supports multiple measurement tasks to help operators analyze network status more comprehensively. Generally, traffic statistics are flow-based, and flow is identified by 5-tuple (i.e., source IP address, destination IP address, source port, destination port and Protocol) or 2-tuple (i.e., source IP address and destination IP address). The flow size is defined as the total bytes of the flow or the number of packets belonging to the flow. This paper discusses the following flow-based measurement tasks:

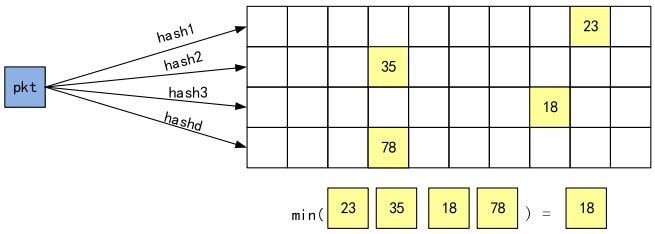

Fig. 1. The data structure of Count-Min sketch.

- Flow size estimation: the number of packets or total byte counts of flows during a time interval.
- Heavy hitters Detection: identify flows whose flow sizes are larger than the threshold during a time interval.
- Heavy changers Detection: identify flows whose changes of flow sizes are larger than the threshold during two adjacent time intervals.
- Superspreader Detection: source hosts whose numbers of distinct destination hosts are larger than the threshold during a time interval.
- DDoS attack Detection: destination hosts whose numbers of distinct source hosts are larger than the threshold during a time interval.

There are only a few heavy hitters in the network, but they account for most of the total traffic. And these heavy flows are very important for network billing and traffic engineering. For example, if they are routed wrongly, it is easy to cause network congestion. In addition, real-time and accurate detection of suspicious flows, such as heavy changers, DDoS attacks, etc., is of great significance to network security and network management. Heavy changers indicate that the network is unstable and abnormal. Superspreaders are potential worm spreaders. DDoS attacks make the target server overwhelmed by numerous zombie hosts and fail to provide services to ordinary users. Compared to heavy-hitter detection or heavy changer detection [29–31], there are only few research works which use sketches to detect superspreaders and DDoS attacks. OpenSketch [6] uses CM sketch, bitmap [32], and reversible sketches [33] to track superspreaders/ DDoS attacks. SpreadSketch [5] combines CM sketch and multi-resolution bitmap to detect superspreaders.

2.2. Sketch

The sketch is a summary data structure that compresses massive data streams into a small space through hash operations. It is an approximate measurement method with high accuracy. Since this concept was introduced, many sketches have been proposed to perform various measurement tasks. As shown in Fig. 1, CM sketch, a typical sketch algorithm, is composed of a two-dimensional array with d rows and w columns. Each row is related to an independent hash function, and the array is initially set to 0. It has two basic operations: update and query. The update algorithm is as follows: For an incoming packet, it is mapped into multiple counters. Each counter increases by the size of the packet. CM sketch uses the d hash functions for the query algorithm to find the corresponding counter first. Then it reports the minimum value among d counters as the flow size. Although CM has been optimized, there are many overestimation errors caused by hash collisions. CU sketch adopts a conservative update strategy for CM sketch. Every time it only updates the smallest of the counters. The other counters are set to the maximum between their original value and the smallest value of the smallest counter. CU sketch reduces false

positives greatly in the heavy hitter detection. Both CM sketch and CU sketch save memory by only storing flow sizes not storing flow identifiers, but heavy hitter detection and heavy changer detection need to track suspicious flows. In this case, these sketches add min-heap structures to keep the flow identifiers of heavy hitters or heavy changers.

2.3. Algorithms for finding frequent elements

The flow size distribution of network traffic is highly skewed, which means that most flows are mouse flows, while only a small percentage of flows are elephant flows. The byte counts of elephant flows occupy the majority of all flows, while the byte counts of mouse flow occupy the minority of all flows. When an elephant flow and a mouse flow are mapped into the same counter, the sketching algorithm reports the total sum of the counter as flow size. So mouse flow is significantly overestimated. Some solutions are proposed to mitigate the problem by separating elephant flows, and mouse flows.

Misra and Gries' 82 [34] is a well-known algorithm for frequent elements mining. Instead of keeping a counter for each flow, the Misra–Gries algorithm only uses a few counters to store flows. The algorithm first judges which counter it belongs to for a new packet. If its information has already existed in the counter, the corresponding counter increases by the packet size; otherwise, the counter subtracts the packet size. During this process, old flows with counters equal to 0 would be replaced by new flows continually. The remaining flows of counters are elephant flows. SketchLearn applies the Misra–Gries algorithm for separating elephant flows. Elastic sketch proposes Ostracism to keep elephant flows in the heavy part, and mouse flows in the light part. The algorithm decides which flows are evicted from the heavy part by voting. HeavyKeeper adopts a new strategy named count-with-exponential-decay to filter mouse flows. Unlike the Misra–Gries algorithm, the exponential decay strategy makes elephant flows decay slowly while making mouse flows decay sharply. Cold Filter [35] uses a two-layer sketch to store elephant flows, and mouse flows, respectively. It defines a threshold in advance to find large flows. MV-Sketch regards every packet as a vote and uses the Majority Vote Algorithm (MJRTY) to track heavy flows. These existing methods separate large flows and small flows very well, but there are still some problems to be solved. Firstly, most separation algorithms make no maximal use of memory. They only save memory usage by not storing the flow IDs of the mouse flows. Secondly, they are not invertible. If some packets which belong to the elephant flow are discarded, they would not be recounted. The problem causes underestimation errors.

3. Tree sketch framework

3.1. Basic structure

3.1.1. Key idea

Our goal is to use sketches to achieve accurate network measurement in limited memory. It is well-known that network traffic is heavy-tailed [36–38]. When a large flow and a small flow are hashed to the same bucket, there are some errors because the small flow is overestimated greatly. Sketch-based algorithms can reduce errors by separating elephant flows and mouse flows. In this paper, Tree sketch makes three improvements to optimize separation operations. Firstly, Tree sketch allocates smaller memory for each mouse flow and larger memory for each large flow. It enhances memory efficiency. Secondly, Tree sketch diminishes the Heavy Cuckoo algorithm's false positives of heavy flow detection. Thirdly, the framework is divided into three substructures. The light part can filter out most of the small flows and avoid complex operations in the heavy part. It achieves a higher processing speed.

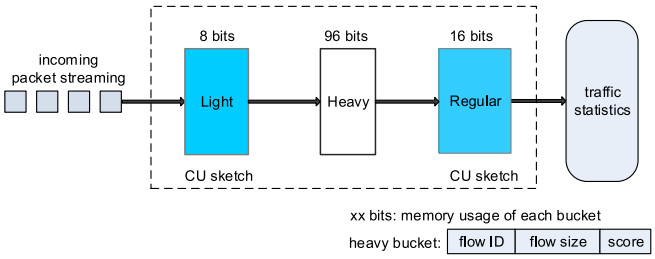

Fig. 2. The data structure of Heavy Cuckcoo.

3.1.2. Heavy Cuckcoo algorithm

A Tree sketch is composed of three branches, and each branch performs different measurement tasks. In this section, we focus on the branch for heavy-hitter detection and heavy changer detection. The other branches are similar to it. As shown in Fig. 2, the structure consists of three parts: heavy part, regular part, and light part. The light part is a CU sketch with r rows and w columns. Each row is associated with a hash function. It updates the smallest counters for each incoming packet. The heavy part is a hash table for recording elephant flows. Each heavy bucket has two fields: flow ID and flow size. The flow ID is defined as source–destination address pairs, and the flow size is defined as the number of packets in this paper. Besides, the regular part is made of a CU sketch and stores the flows filtered from the heavy part. The number of buckets is much less than that in the light part. After processing all packets, the Tree sketch gets the corresponding statistics from each part. For example, the light part is for cardinality estimation, and the heavy part is for heavy-hitter detection.

Insert Operation: Heavy cuckoo is the core algorithm of Tree sketch. It separates elephant flows from mouse flows and stores the real elephant flows in the heavy buckets. In this section, we describe the algorithm in detail.

Algorithm 1 shows the insert operation of Heavy Cuckoo. The algorithm first queries its minimum in the light sketch for each incoming packet. If the minimum is less than the threshold, it updates the packet in the light part; If not, it updates the packet in the heavy part. We define the number of packets as flow size in this paper. When the packet f is sent to the heavy part, we map it to the first bucket $B_1(f_1, v_1)$. If B_1 is not empty or f_1 is not equal to f, the packet will be mapped into the second bucket $B_2(f_2, v_2)$. Some packets of flow f may be inserted into the regular part before the packet f arrives. In this algorithm, these packets are named R. There are five cases in the heavy buckets.

(1) B_1 is empty or $f = f_1$, the algorithm updates B_1 by $(f, 1)$.
(2) $f \neq f_1$, but B_2 is empty or $f = f_2$, the algorithm inserts f into B_2.
(3) $f \neq f_1$ and $f \neq f_2$, there are hash collisions in both B_1 and B_2. We firstly query the flow size of f in the regular part. Then the algorithm compares f_1 with R. If $R > v_1$, the flow (f_1, v_1) will be evicted to the regular part. The new flow f is inserted into B_1 by $(R + 1)$.
(4) Neither f_1 nor f_2 is equal to f. $R \leqslant v_1$ but $R > v_2$, the algorithm updates B_2 by $(R + 1)$.
(5) $f \neq f_1$ and $f \neq f_2$, $R \leqslant v_1$ and $R \leqslant v_2$, the flow of f is small and there is no replace between B_1 and B_2. The algorithm inserts new packet into the regular part directly.

Query Operation: The query algorithm firstly finds the smallest counter of flow f in the light part (Algorithm 2). If the minimum is less than the light part threshold, the flow is only stored in the light part. So the algorithm reports the minimum as the flow size. If the minimum is less than the threshold, the excess of the flow will be sent to the heavy part or the regular part. If the flow ID of B_1 or B_2 equals f, the flow exists in the heavy part, and flow size is the sum of two parts (light part and

Algorithm 1 Heavy Cuckoo Update Algorithm

Require: packet(f, v_f)
procedure INSERT(f, v_f)
  $l \leftarrow min(L[h_i(f)])$
  if $l \leq \tau$ **then**
    $CU_L(f, v_f)$
  else
    HEAVY(f, v_f)
  end if
end procedure
function HEAVY(f, v_f)
  if $B_1[h_i(f)].key = f$ **then**
    $B_1[h_i(f)].size \leftarrow size + v_f$
  else if $B_2[h_j(f)].key = f$ **then**
    $B_2[h_j(f)].size \leftarrow size + v_f$
  else
    $r \leftarrow min(R[h_i(f)])$
    if $r > B_1[h_i(f)].size$ **then**
      $CU_R(B_1[h_i(f)].key, B_1[h_i(f)].size)$
      $B_1(key, size) \leftarrow (f, v_f + r)$
      $CU_R(f, -r)$
    else if $r > B_2[h_i(f)].size$ **then**
      $CU_R(B_2[h_j(f)].key, B_2[h_j(f)].size)$
      $B_2(key, size) \leftarrow (f, v_f + r)$
      $CU_R(f, -r)$
    else
      $CU_R(f, v_f)$
    end if
  end if
end function
function $CU_i(f, v_f)$
  $cu \leftarrow min(CU_R[h_i(f)])$
  for $i = 1$ to d **do**
    if $\min(CU_R[h_i(f)])]) = cu$ **then**
      $CU_R[h_i(f)]] \leftarrow cu + v_f$
    else
      $CU_R[h_i(f)] \leftarrow max(CU_R[h_i(f)], cu + v_f)$
    end if
  end for
end function

heavy part); otherwise, the algorithm returns the sum of the light part and regular part.

Similar to cuckoo hashing, the Heavy Cuckoo algorithm uses two hash functions to improve the space utilization of the hash table. For an incoming packet, it has three positions, B_1, B_2 from the heavy part, and R from the regular part. The algorithm can keep two larger flows of them in the heavy part. However, there are lots of hash collisions if the hash table is filled up. The flow size distribution is highly skewed. The light part stores most of the flows, and only a few flows are sent to the heavy part. This reduces the probability of hash collisions greatly.

3.2. Tree sketch

3.2.1. Data structure

As mentioned above, a Tree sketch consists of three basic data structures. Each branch is designed for different tasks, but the structure is slightly different. As shown in Fig. 3, the Tree sketch is divided into three parts: light part for small flows, heavy part for large flows, and regular part for remaining flows. The light part has a bloom filter and three CU sketches. Bloom filter can identify whether the current flow has existed in the Tree sketch. The CU sketch for heavy-hitter detection has the same hash functions to bloom filter [39]. The hash

Algorithm 2 Heavy Cuckoo Query Algorithm

Require: packet(f)
Ensure: s_f
$l \leftarrow min(L[h_i(f)])$
if $l \leq \tau$ **then**
  $s_f \leftarrow l$
else if $H[h_i(f)].key == f$ **then**
  $s_f \leftarrow H[h_i(f)].size + \tau$
else
  $s_f \leftarrow R[h_i(f)] + \tau$
end if

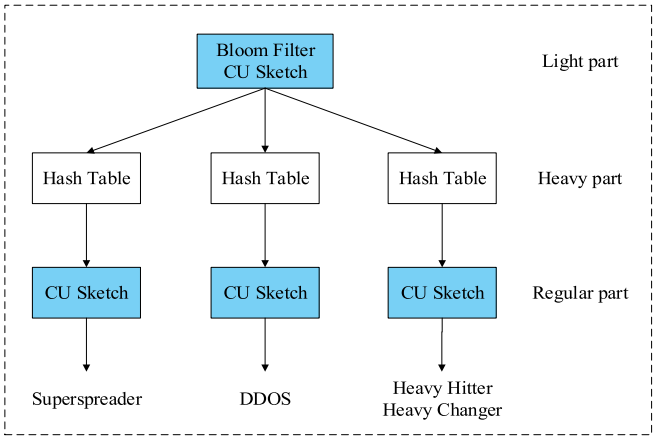

Fig. 3. The data structure of Tree sketch.

results can be used in two structures at the same time. The hash results can be used in two structures simultaneously to accelerate packet processing. The heavy part is composed of three independent hash tables, in which each bucket has a key–value pair. The regular part has three CU sketches for storing flows evicted from the heavy part. Besides, superspreader detection and DDoS attack detection focus on the flow. Heavy hitter detection and heavy changer detection focus on the packet. The magnitude is different, so we allocate a suitable memory for them.

3.2.2. Measurement tasks

Given an incoming packet, the algorithm first exacts its source and destination addresses as flow ID. Then Tree sketch maps it into bloom filter and CU sketch, respectively, according to the hash value. Bloom filter judges whether the flow is in the Tree sketch. CU sketch queries the minimum at the same time.

Heavy Hitter Detection: After all packets finish insert operations, we iterate over the hash table for heavy-hitter detection. If the sum of the heavy and light parts is more than the predefined threshold, we report this flow as a heavy hitter.

Heavy Changer Detection: We build two Heavy Cuckoo structures (named H1 and H2 respectively) to store statistics during two adjacent intervals. Firstly, the algorithm checks all flow in the heavy part of H1. If the flow size is larger than T, the algorithm computes the difference between H1 and H2. Then heavy changer list adds the flows whose difference is larger than T. The algorithm checks all flows in H2. If the flow size is larger than T and not in the list, the Tree sketch will count the difference. If its difference is larger than T, the flow will be added to the list. At last, the list has all heavy changers.

Superspreader Detection: When the bloom filter judges that the packet belongs to a new flow, the algorithm exacts the source IP address from 2-tuple, and the corresponding components count the number of the flow. If its size is larger than the threshold, the algorithm regards the flow as a superspreader.

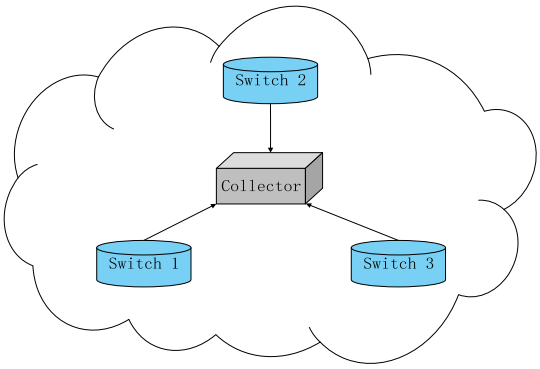

Fig. 4. Network-wide measurements. The collector integrates information from multiple measurement nodes regularly.

DDoS Detection: When the bloom filter judges that the packet is not in Tree sketch, the algorithm exacts the destination IP address from 2-tuple as a new flow ID. The structure for DDoS attack detection counts the distinct source IP address for every destination IP address. After all flows are processed, the algorithm reports the flow as a DDoS attack if its size exceeds the threshold.

3.3. SIMD

SIMD is a parallelism technology that simultaneously performs the same operation on multiple data points. We use SIMD instructions to perform all hash operations in the light and regular parts. The flow ID is divided into t parts. Each part is regarded as a new flow ID. And these new flow IDs are mapped to buckets by SIMD. It can perform r hash functions in parallel. So the technology boosts the performance of Tree Sketch.

3.4. Netwide-wide measurement

Single node measurement cannot meet the requirement in accuracy [13]. For example, a flow belongs to the mouse flow in a single node. Actually, it is an elephant flow because its flow size exceeds the threshold in the entire network. So Tree sketch needs to acquire more accurate results by network-wide measurement. As shown as Fig. 4, Tree sketch integrates information from multiple measurement nodes. If their flow IDs are the same, the algorithm merges these flows. If these flows are larger than the threshold, Tree sketch reports them as final heavy flows.

4. Experiments

4.1. Experiment setup

Testbed: We run all the experiments on a server with 18-core CPUs (2 threads, Intel(R) Core(TM) i9-9980XE CPU @ 3.00 GHz) and 32 GB total memory. The server runs Ubuntu 18.04. Each CPU has 64 KB of L1 cache, 1024 KB of L2 cache, and 24MB of L3 cache. To exclude the I/O overhead, Tree sketch will allocate 500MB buffers to store all traces.

Datasets: We use CAIDA's anonymized Internet traces captured on 10GE link monitor city in January 2019 [40]. We divide each trace into 5s-long subtraces. Each subtrace is 220MB, which contains 3M packets and 220 K flows.

Metrics:

- Average Absolute Error (AAE): $\frac{1}{n}\sum_{i=1}^{n}|\hat{v}_i - v_i|$, where $\hat{v}_i$ is the estimated size of the flow f_i, v_i is the ground-truth size of f_i, n is the number of distinct flows. AAE measures the average magnitude of the sum of errors.

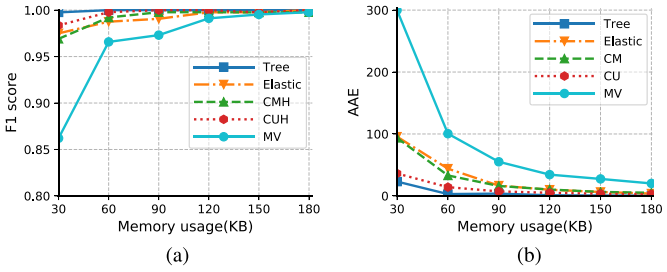

Fig. 5. (a) F1 score for heavy hitter detection; (b) AAE for heavy hitter detection.

- Average Relative Error (ARE): $\frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{v}_i - v_i|}{v_i}$. Similar to AAE, ARE evaluates the accuracy of measurement tasks.
- F1-score: $\frac{2*precision*recall}{precision+recall}$, where *precision* is the ratio of true instances among reported instances and *recall* is the ratio of reported instances among all true instances. F1 score is the harmonic mean of precision and recall where the closer the F1 score towards one, the better the result.
- Throughput: $\frac{N}{T}$, where N denotes the number of packets and T is the process time. It represents the number of packets processed per second (pkts/s).

Parameters: As mentioned in Section 3, the Tree sketch comprises three branches. Each branch sets an 8-bit counter for the light part, a 32-bit counter for the heavy part, and a 16-bit counter for the regular part. And the width and depth of the sketch are decided by allocated memory. Moreover, there are five thresholds in the Tree sketch. The threshold of the light part is 255. If its flow size is no less than 255, the packet would be sent to the heavy part to process. The threshold for heavy changer detection is set to a fixed size. The value is 5000 in this paper. The threshold for heavy-hitter detection is defined as the fraction of total packets (such as 0.2%). The threshold for superspreader and DDoS attack detections are 0.2% and 0.18% of total flows respectively. The experiments use MurmurHash for all algorithms. Note that all algorithms set the depth of every sketch as four except the Elastic sketch. Both heavy part and light part in Elastic sketch set one hash function.

Benchmarks: In this paper, we compare Tree sketches to state-of-the-art solutions for different measurement tasks. Tree sketch (Tree) is compared to four other algorithms for heavy-hitter detection and heavy changer detection. They are Elastic sketch, CM sketch with minHeap (CMH), CU sketch with minHeap (CUH), and MV-Sketch (MV), respectively. For superspreader detection and DDoS attack detection, the experiment compares Tree, CMH, CUH, and SpreadSketch (Spread). These algorithms share and uses the same hash functions.

4.2. Accuracy

Experiment 1: Fig. 5 compares the accuracy of five algorithms for detecting heavy hitters. When the memory usage is 30 KB, The F1 score of Tree sketch is 0.997, which is larger than other sketch algorithms. And Tree sketch can achieve 100% precision and recall with 60 KB memory size. When memory size is 30 KB, the AAE of five algorithms is 23, 95, 93, 36, and 299, respectively. Besides, Fig. 5(b) shows that the AAE of the Tree sketch always keeps the smallest in a different memory. Therefore, Tree sketch is much more accurate for heavy-hitter detection than the other algorithms in small memory.

Experiment 2: Fig. 6 compares the accuracy of five algorithms for heavy changer detection. When the memory size is 30 KB, MV-Sketch's F1 score is only 0.47. The F1 score of Tree sketch and Elastic sketch is larger than the other three algorithms. The AAE of Tree sketch is always smaller than the other four algorithms when the memory size is from 30 KB to 180 KB. Tree sketch can achieve higher accuracy for heavy changer detection.

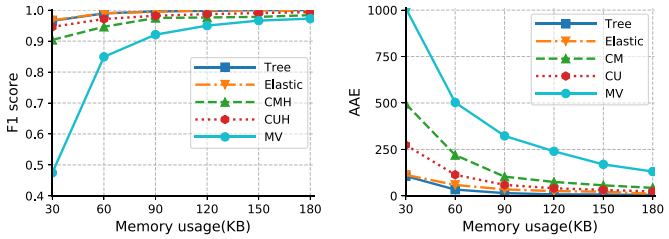

Fig. 6. (a) F1 score for heavy changer detection; (b) AAE for heavy changer detection.

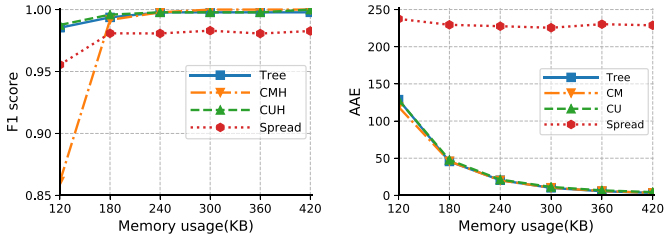

Fig. 7. (a) F1 score for superspreader detection; (b) AAE for superspreader detection.

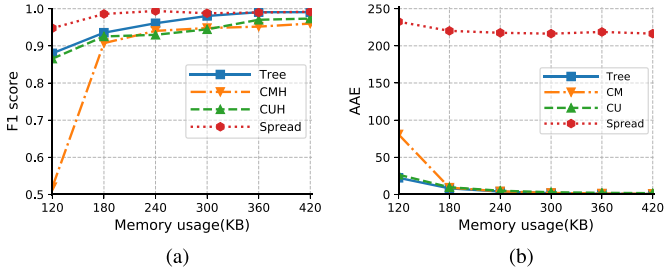

Fig. 8. (a) F1 score for DDoS attack detection; (b) AAE for DDoS attack detection.

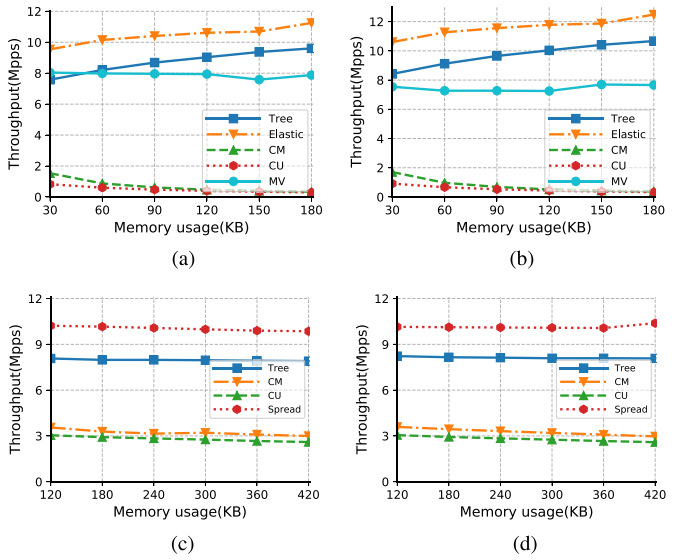

Fig. 9. (a) Throughput for heavy hitter detection; (b) Throughput for heavy changer detection; (c) Throughput for superspreader detection; (d) Throughput for DDoS attack detection.

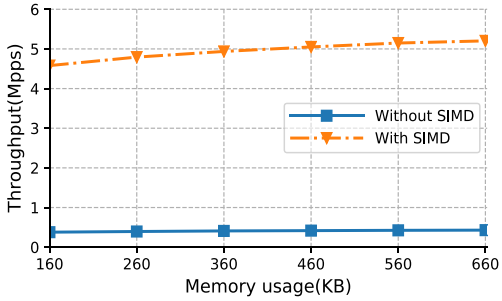

Fig. 10. Performance optimizations of Tree sketch.

Experiment 3: Fig. 7 compares the accuracy of Tree, CMH, CUH, and Spread in superspreader detection. The F1 score of Tree and CUH are all 0.98 in 120 KB memory and 1.0 in 420 KB memory. They are always larger than the other three algorithms. Fig. 7(b) shows that the AAE of Tree, CMH, and CUH is smaller than Spread. Therefore, we can conclude that Tree sketch can achieve accurate superspreader detection.

Experiment 4: Fig. 8 compares the accuracy of Tree, CMH, CUH, and Spread in DDoS attack detection. The F1 score and AAE of Tree are 0.99 and 1.01, respectively when the memory size is 360 KB. With the same memory, the F1 score of Spread is also 0.99, but its AAE is 218.7. Tree's F1 score is larger than the F1 score of CMH and CUH, and Tree's AAE is smaller than the AAE of CMH and CUH. For DDoS attack detection, Tree sketch achieves higher accuracy than other algorithms.

4.3. Throughput

Experiment 5: Figs. 9(a) and 9(b) show the throughput of Tree, Elastic, CMH, CUH, and MV in heavy hitter detection and heavy changer detection. All algorithms maintain the same parameter settings, including threshold and memory usage. When the memory size is from 30 KB to 180 KB, tree achieves the highest throughput among the five algorithms. In particular, when the memory size is 180 KB, the throughput of Tree is 32 times and 37 times of CMH and CMU, respectively. As the memory increases, the throughput of Tree and Elastic continues to increase, yet the other three algorithms suffer from a drop in throughput. When the memory size increases, Tree and Elastic have less hash collision in the heavy part, avoiding other operations in the regular or light parts. Nevertheless, the other three algorithms need more comparison operations or sort operations, so their throughput decreases.

Figs. 9(c) and 9(d) show the throughput of four algorithms in superspreader detection and DDoS attack detection. The tree has a higher throughput than CMH and CUH. Compared to Spread, Tree increases a Bloom Filter to find distinct flows. So the throughput of Tree's throughput is slightly less than Spread's. Besides, four algorithms have a throughput drop as the memory size increases. That is because sketches need to find superspreaders or DDoS attacks from more buckets. These lookup operations cause a decrease in throughput.

Experiment 6: Fig. 10 compares the throughput of Tree between the original version and the optimized version. The optimized version uses SIMD instructions to speed up packet processing. When the memory size is from 160 KB to 660 KB, the throughput of Tree is from 378228 to 429443 without SIMD. After applying SIMD in Tree, its throughput is from 4580080 to 5203170 with SIMD. The SIMD version achieves 11 higher throughputs than the original version. The results show that the SIMD instructions greatly accelerate Tree's processing speed.

4.4. Network-wide measurement

Experiment 7: Fig. 11(a) shows the recall of Tree and CMH in four measurement tasks, which the measurement results are integrated from three measurement points. Tree and CMH set the same threshold and

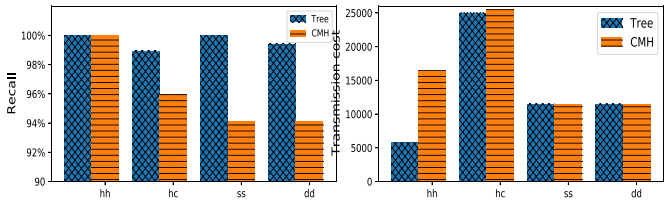

Fig. 11. (a) Recall for network-wide measurement; (b) Transmission cost for network-wide measurement.

memory size to obtain reliable results. In heavy hitter detection, the threshold in the collector is set as 0.25% of the total number of packets. The threshold in each node is set as 0.20% of the number of its packets. Both Tree and CMH achieve 100% recall. In heavy changer detection, the recall of Tree and CMH are 98.95% and 95.99%, respectively. In superspreader detection, the Tree has 100% recall, and CMH has 94.12% recall. The Tree-based method can achieve 99.41%, and CMH achieves 94.12% recall in DDoS attack detection. Overall, Tree has a higher recall than CMH in three measurement nodes. Fig. 11(b)shows the transmission cost of Tree and CMH in four measurement tasks. In this paper, transmission cost is defined as the total amount of data transmitted from each measurement point to the collector. Since the capacity of a network communication link is limited, the less transmission cost for network measurement, the more bandwidth for other network data. For heavy hitter detection, the transmission cost of Tree and CMH are 5788.8 bits and 16416 bits across three measurement nodes. Moreover, Tree achieves a slightly lower transmission cost than CMH in heavy changer detection, in which the cost of Tree is 25204 bits, and the cost of CMH is 25546 bits. Tree and CMH have the same transmission cost for detecting superspreaders and DDoS attacks. Overall, Tree has a lower transmission cost than CMH.

5. Conclusion

This paper introduced a novel method, Tree sketch, a memory-efficient data structure for traffic measurement. The heavy cuckoo algorithm effectively reduces hash collisions by separating elephant and mouse flows. The Tree sketch also uses the skewed distribution of network traffic, which stores flow in different categories, improving the utilization of space greatly. The framework shows how to update, and query flows to perform different measurement tasks, such as heavy hitter detection and heavy changer detection. The Tree sketch also applies SIMD instructions to accelerate packet processing. Moreover, the Tree sketch is extensible and could add other measurement functions without extra hardware overhead, like cardinality estimation and flow size distribution. The experimental results show that Tree sketch achieved more accurate traffic measurement in small memory.

CRediT authorship contribution statement

Lei Liu: Conceptualization, Methodology, Formal analysis, Writing – original draft, Writing – review & editing. **Tong Ding:** Model design, Software, Writing – original draft. **Hui Feng:** Formal analysis, Investigation, Writing – review & editing. **Zhongmin Yan:** Data curation, Validation. **Xudong Lu:** Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by NSFC (No. 61972230) and NSFShandong (No. ZR2021LZH006).

References

[1] X. Zhang, D. Wang, K. Ota, M. Dong, H. Li, Exponential stability of mixed time-delay neural networks based on switching approaches, IEEE Trans. Cybern. 52 (2) (2022) 1125–1137, http://dx.doi.org/10.1109/TCYB.2020.2985777.

[2] M. Tao, K. Ota, M. Dong, DSARP: Dependable scheduling with active replica placement for workflow applications in cloud computing, IEEE Trans. Cloud Comput. 8 (4) (2020) 1069–1078, http://dx.doi.org/10.1109/TCC.2016.2628374.

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

[4] G. Cheng, Y. Tang, Line speed accurate superspreader identification using dynamic error compensation, Comput. Commun. 36 (13) (2013) 1460–1470.

[5] L. Tang, Q. Huang, P.P. Lee, SpreadSketch: Toward invertible and network-wide detection of superspreaders, in: Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 1608–1617.

[6] M. Yu, L. Jose, R. Miao, Software defined traffic measurement with OpenSketch, in: Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, 2013, pp. 29–42.

[7] S. Ganguly, M. Garofalakis, R. Rastogi, K. Sabnani, Streaming algorithms for robust, real-time detection of ddos attacks, in: Proceedings of the 27th International Conference on Distributed Computing Systems, ICDCS'07, IEEE, 2007, p. 4.

[8] H. Li, K. Ota, M. Dong, Virtual network recognition and optimization in SDN-enabled cloud environment, IEEE Trans. Cloud Comput. 9 (2) (2021) 834–843, http://dx.doi.org/10.1109/TCC.2018.2871118.

[9] G. Min, J. Hu, M.E. Woodward, Performance modelling and analysis of the TXOP scheme in wireless multimedia networks with heterogeneous stations, IEEE Trans. Wireless Commun. 10 (12) (2011) 4130–4139.

[10] H. Li, K. Ota, M. Dong, Deep reinforcement scheduling for mobile crowdsensing in fog computing, ACM Trans. Internet Technol. 19 (2) (2019) 1–18.

[11] J. Hu, G. Min, W. Jia, M.E. Woodward, Comprehensive QoS analysis of enhanced distributed channel access in wireless local area networks, Inform. Sci. 214 (2012) 20–34.

[12] H. Li, K. Ota, M. Dong, LS-SDV: Virtual network management in large-scale software-defined IoT, IEEE J. Sel. Areas Commun. 37 (8) (2019) 1783–1793.

[13] R. Harrison, Q. Cai, A. Gupta, J. Rexford, Network-wide heavy hitter detection with commodity switches, in: Proceedings of the Symposium on SDN Research, 2018, pp. 1–7.

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

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

[16] Y. Fu, D. Li, S. Shen, Y. Zhang, K. Chen, Clustering-preserving network flow sketching, in: Proceedings of the IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 1309–1318.

[17] G. Cormode, S. Muthukrishnan, What's new: Finding significant differences in network data streams, IEEE/ACM Trans. Netw. 13 (6) (2005) 1219–1232.

[18] P. Roy, A. Khan, G. Alonso, Augmented sketch: Faster and more accurate stream processing, in: Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 1449–1463.

[19] Y. Liu, W. Chen, Y. Guan, A fast sketch for aggregate queries over high-speed network traffic, in: Proceedings of the 2012 IEEE INFOCOM, IEEE, 2012, pp. 2741–2745.

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

[21] P. Xiao, W. Qu, H. Qi, Y. Xu, Z. Li, An efficient elephant flow detection with cost-sensitive in SDN, in: Proceedings of the 2015 1st International Conference on Industrial Networks and Intelligent Systems, INISCom, IEEE, 2015, pp. 24–28.

[22] G. Cormode, S. Muthukrishnan, Summarizing and mining skewed data streams, in: Proceedings of the 2005 SIAM International Conference on Data Mining, SIAM, 2005, pp. 44–55.

[23] R. Ben-Basat, G. Einziger, R. Friedman, Y. Kassner, Heavy hitters in streams and sliding windows, in: Proceedings of the IEEE INFOCOM 2016-the 35th Annual IEEE International Conference on Computer Communications, IEEE, 2016, pp. 1–9.

[24] Y. Zhou, O. Alipourfard, M. Yu, T. Yang, Accelerating network measurement in software, Acm Special Interest Group Data Commun. 48 (3) (2018) 2–12.

[25] L. Tang, Q. Huang, P.P.C. Lee, MV-Sketch: A fast and compact invertible sketch for heavy flow detection in network data streams, in: Proceedings of the IEEE INFOCOM 2019 - IEEE Conference on Computer Communications, 2019, pp. 2026–2034.

[26] J. Gong, T. Yang, H. Zhang, H. Li, S. Uhlig, S. Chen, L. Uden, X. Li, HeavyKeeper: An accurate algorithm for finding top-k elephant flows, in: Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference, 2018, pp. 909–921.

[27] Q. Huang, P.P.C. Lee, Y. Bao, Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference, in: Proceedings of the ACM Special Interest Group on Data Communication, 2018, pp. 576–590.

[28] B. Fan, D.G. Andersen, M. Kaminsky, M.D. Mitzenmacher, Cuckoo filter: Practically better than bloom, in: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies, 2014, pp. 75–88.

[29] B. Turkovic, J. Oostenbrink, F.A. Kuipers, Detecting heavy hitters in the data-plane, 2019, arXiv: Networking and Internet Architecture.

[30] M. Hamdan, B. Mohammed, U. Humayun, A. Abdelaziz, S. Khan, M.A. Ali, M. Imran, M.N. Marsono, Flow-aware elephant flow detection for software-defined networks, IEEE Access 8 (2020) 72585–72597.

[31] Y. Li, R. Miao, C. Kim, M. Yu, FlowRadar: a better NetFlow for data centers, in: Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation, 2016, pp. 311–324.

[32] C. Estan, G. Varghese, M. Fisk, Bitmap algorithms for counting active flows on high-speed links, IEEE ACM Trans. Netw. 14 (5) (2006) 925–937.

[33] R.T. Schweller, A. Gupta, E. Parsons, 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, 2004, pp. 207–212.

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

[35] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, 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, 2018, pp. 741–756.

[36] C. Estan, G. Varghese, New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice, ACM transactions on computer systems 21 (3) (2003) 270–313.

[37] P. Roy, A. Khan, G. Alonso, Augmented sketch: Faster and more accurate stream processing, in: Proceedings of the 2016 International Conference on Management of Data, 2016, pp. 1449–1463.

[38] R. Cohen, Y. Nezri, Cardinality estimation in a virtualized network device using online machine learning, IEEE ACM Trans. Netw. 27 (5) (2019) 2098–2110.

[39] A. Goel, P. Gupta, Small subset queries and bloom filters using ternary associative memories, with applications, ACM SIGMETRICS Performance Evaluation Review 38 (1) (2010) 143–154.

[40] CAIDA, The CAIDA anonymized internet traces dataset, https://www.caida.org/data/passive/passive_dataset.xml.