One Memory Access Sketch: a More Accurate and Faster Sketch for Per-flow Measurement

Yang Zhou[1], Peng Liu[1], Hao Jin[1], Tong Yang[1,2], Shoujiang Dang[3,4], Xiaoming Li[1]
Department of Computer Science and Technology, Peking University, China[1]
Collaborative Innovation Center of High Performance Computing, NUDT, China[2]
National Network New Media Engineering Research Center, Institute of Acoustics,
Chinese Academy of Sciences, China[3], University of Chinese Academy of Sciences, China[4]

***Abstract*—Sketch is a probabilistic data structure widely used for per-flow measurement in the real network. The key metrics of sketches for per-flow measurement are their memory usage, accuracy, and speed. There are a variety of sketches, but they cannot achieve both high accuracy and high speed at the same time given a fixed memory size. To address this issue, we propose a new sketch, namely the OM (One Memory) sketch. It achieves much higher accuracy than the state-of-the-art, and achieves close to one memory access and one hash computation for each insertion or query. The key methodology of our OM sketch is to leverage word constraint and fingerprint techniques based on a hierarchical structure. Extensive experiments based on real IP traces show that the accuracy is improved up to 10.64 times while the speed is improved up to 2.50 times, compared with the well-known CM sketch [1]. All the related source code has been released at GitHub [2].**

I. Introduction

A. Background and Motivation

As the modern society is entering into a big network data era, one fundamental issue is how to monitor and manage the network efficiently or so-called network management. Within this grand issue, one of the most significant aspects is the per-flow measurement, which refers to estimating the *number of packets in each flow* during a time period. One flow is uniquely identified by its flow ID extracted from the packet header or packet content. For example, we have per-source flows, per-destination flows, TCP flows which consist of 5-tuples and even flows based on their packet content. Per-flow measurement has many important applications. For example, by estimating the number of packets in each TCP flow, ISPs can devise suitable strategies for traffic routing to alleviate the network congestion [3]. By estimating the number of SYN packets sent from each source IP, network administrators can detect the SYN flooding attack timely [4], [5]. In SDN networks, per-flow measurement can help the network administrators to intelligently allocate network bandwidth among different paths. The intrusion detection systems can use the information from per-flow measurement to find heavy hitters [6] and to monitor the heavy changes happening in certain paths [7], [8]. As per-flow measurement plays a significant role in the modern network management, many algorithms have been proposed [9]–[11].

One kind of effective solutions for per-flow measurement is based on a probabilistic data structure called *sketch* [12]–[14], a two dimensional array. The sketch has two primary operations: insertion and query. During insertions, it records the information about flow sizes compactly. During queries, it reports the corresponding flow sizes given flow IDs. The key mechanism of the sketch is called **counter-sharing**, which means embracing the hash collisions and letting two or more flows share the same counter(s). In conventional sketches, the counter size needs to be specially tailored in order to accommodate the maximum flow size. However, the flow size distribution in the real network is highly *skewed* [15], [16]. In other words, a few flows have large sizes (*e.g.*, > 40000) while plenty of flows have small sizes (*e.g.*, < 16) (they are called *elephant flows and mice flows*, respectively). Therefore, the high-order bits in most counters of conventional sketches are wasted. This kind of memory inefficiency will inevitably degrade the accuracy. Besides, the conventional sketches cannot perfectly catch up with the fast line speed of network flows because it needs three or more memory accesses and hash computations for each insertion or query. The design goal of this paper is to *devise a new sketch which achieves high accuracy, high insertion speed and high query speed at the same time*.

B. Prior Art

There are three typical sketches widely used for per-flow measurement: the CM sketch [1], the CU sketch [17] and the Count sketch [18]. A CM sketch consists of d arrays: $A_1, A_2, ..., A_d$, each of which consists of w counters and corresponds with a hash function $h_i(.) (1 \leqslant h_i(.) \leqslant w, 1 \leqslant i \leqslant d)$. When inserting a flow e, the CM sketch first computes the d hash functions and locates the d counters: $A_1[h_1(e)]$, $A_2[h_2(e)]$, ..., $A_d[h_d(e)]$. Then it increments all the d **hashed counters**. When querying a flow e, the CM sketch reports the minimum value of the d hashed counters. The CU sketch [17] makes a slight but effective modification, namely conservative update, compared with the CM sketch. During insertions, the CU sketch only increases the smallest counter(s) (there may be two or more smallest counters) among the d

*Corresponding author: Tong Yang (Email: yang.tong@pku.edu.cn). This work was done by Yang Zhou, Peng Liu, and Hao Jin under the guidance of their mentor: Tong Yang. This work is supported by National Basic Research Program of China (2014CB340400).

hashed counters while the query process keeps unchanged. The Count sketch [18] is also similar to the CM sketch except that each array uses an additional hash function to smooth the accidental errors. Among these sketches, the CU sketch achieves the best performance in terms of both accuracy and speed. Unfortunately, as mentioned in section I-A, all the above sketches suffer from the memory inefficiency, multiple memory accesses, and multiple hash computations for each insertion or query. In other words, they all have the following shortcomings: *1) the accuracy is poor when using small memory; 2) the insertion and query speed is limited.*

C. The Proposed Solution

In this paper, we propose a novel sketch, namely the OM (**O**ne **M**emory) sketch. The key idea of our OM sketch is to *leverage word constraint and fingerprint techniques based on a hierarchical structure, while achieving high accuracy and high speed.* We organize the OM sketch as a hierarchical structure in which the higher layers possess less memory. Specifically, the lower layers with small counter sizes mainly record the information of mice flows and the higher layers with relatively large counter sizes mainly record the information of elephant flows. In this way, the memory efficiency is significantly improved. When one or more counters overflow at the lower layer, we use the higher layer to record its number of overflows. We call it **hierarchical counter-sharing**. Besides, we have the following two critical observations: 1) The counter size of the lower layer is naturally small (*e.g.*, 4bits) which means one machine word can contain enough counters. 2) The elephant flows in the real network are very scarce, which means the probability that the insertion of one arbitrary flow accesses the second or higher layer is very low (*e.g.*, 1/20). Therefore, we can constrain the corresponding hashed counters within one or several machine words and achieve close to one memory access for each insertion. We call it **word acceleration**. To accelerate the hash computation, we leverage the hash bit technique [19] to locate multiple hashed counters within one or several machine words at each layer through a 64-bit hash value. We further improve the accuracy of OM sketch by using the **fingerprint check** technique: we record the fingerprints of the overflowed flows in their corresponding machine words at the lower layer in order to distinguish them from non-overflowed flows during queries.

D. Key Contributions

1) We propose a new sketch named the OM sketch with three key techniques: hierarchical counter-sharing, word acceleration and fingerprint check.

2) We conduct extensive experiments on the real IP traces, and results show that our OM sketch can achieve superior advantages in terms of both accuracy and speed compared with the state-of-the-art.

II. Related Work

Considering the growing significance of per-flow measurement, abundant solutions have been proposed to solve this problem. Those solutions fall into three main categories: the sketches, the counter variants and Bloom filter variants.

Sketches [15]: The CM, CU, and Count sketch have already been introduced in Section I-B.

Counter variants: Counter Braids [20] and Randomized Counter Sharing [11] are two typical algorithms of Counter variants. In Counter Braids, the per-flow measurement requires a post process. Specifically, before querying one certain flow, the information of all distinct flows must be obtained in advance. As a result, the query speed is significantly slowed down. The Randomized Counter Sharing randomly increments one of the d hashed counters during insertions, and reports the sum of all the d hashed counters subtracted by the noise during queries. This algorithm sacrifices its accuracy for high speed.

Bloom filter variants: This kind of solutions are based on Bloom filter [21], which can tell whether a flow belongs to a set or not, but cannot report the flow frequency. Counting Bloom filters (CBF) [22] can estimate the flow frequency. The mechanism of CBF resembles that of the CM sketch, except that CBF only has one array. Based on CBF, some other variants for storing the flow frequency have been proposed: the Spectral Bloom Filter (SBF) [23] and the Dynamic Count Filter (DCF) [24]. However, all these algorithms require multiple memory accesses and hash computations for each insertion or query, which degrades the speed.

To sum up, although various solutions have been proposed in this field, none of the existing algorithms can achieve high accuracy and high speed at the same time.

III. The OM Sketch Algorithm

In this section, we first clarify the reason why we mainly focus on the OM sketch with two layers. Then we present three key techniques of our OM sketch one by one: *hierarchical counter-sharing, word acceleration and fingerprint check.* By leveraging these three techniques, our OM sketch can achieve close to one memory access and one hash computation averagely (three memory accesses and two hash computations in the worst case) for each insertion or query, while improving the accuracy remarkably.

A. Multi-layer OM sketch

The multi-layer OM sketch has γ layers: L_i ($1 \leqslant i \leqslant \gamma$), in order to conduct per-flow measurement at a fine granularity. Each layer L_i has its own counter size δ_i and number of counters w_i. When inserting a flow e, we first update the layer L_1. If one or more counters overflow at the layer L_1, we use the layer L_2 to record the corresponding numbers of overflows. This process continues recursively until there is no overflow(s) at a certain layer. The OM sketch with a large γ (*e.g.*, 3 or more) can achieve a high memory efficiency, but will incur at least γ memory accesses in the worst case. If γ is relatively big (*e.g.*, 3 or more), the OM sketch cannot catch up with the line speed in some cases. Besides, the experiment results show that the OM sketch with three or more layers only improves the accuracy a little compared with the OM sketch with two layers. Therefore, in most cases, we set γ to 2 in order to alleviate the speed decline of the OM sketch in the worst case. In the remainder of the section, we will present the two-layer OM sketch in detail. Note that the mechanism of the multi-layer OM sketch is almost the same as that of the two-layer OM sketch.

B. *Two-layer OM sketch*

1) Hierarchical Counter-sharing:

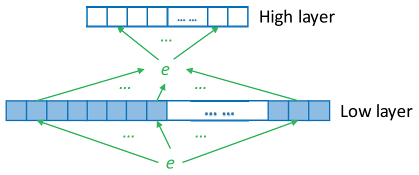

Fig. 1: Basic structure of our OM sketch.

Data structure: As shown in Figure 1, our OM sketch (for convenience, we just refer to the two-layer OM sketch as the OM sketch in the remainder of the paper) consists of two hierarchical layers: the low layer and the high layer. The low layer L_l consists of w_l counters with δ_l *bits* (ranging from 0 to $2^{\delta_l}-1$), and the high layer L_h consists of w_h counters with δ_h *bits* (ranging from 0 to $2^{\delta_h}-1$). We have $\delta_l < \delta_h$ and the reason will be mentioned later. We represent the i^{th} counter of the layer L_l or L_h as $L_l[i]$ or $L_h[i]$. The low layer is associated with d_l independent hash functions $h_l^i(.)(1 \leqslant i \leqslant d_l)$, whose outputs are uniformly distributed in the range $[1, w_l]$, and the high layer is associated with d_h independent hash functions $h_h^i(.)(1 \leqslant i \leqslant d_h)$, whose outputs are uniformly distributed in the range $[1, w_h]$.

There are following two primary operations in our OM sketch: insertion and query. Initially, all the counters of the two layers are set to zero.

Insertion: When inserting a flow e, we first compute the d_l hash functions $h_l^1(e), h_l^2(e), ..., h_l^{d_l}(e)$ $(1 \leqslant h_l(.) \leqslant w_l)$ using the flow ID of e, and locate the d_l *hashed counters* $L_l[h_l^1(e)]$, $L_l[h_l^2(e)]$, ..., $L_l[h_l^{d_l}(e)]$ at the low layer L_l. Then we increment the smallest counter(s) among the d_l hashed counters. When overflows happen at the low layer, we observe that: *the flow e must cause its corresponding d_l hashed counters to overflow concurrently*. For the overflows at the low layer, there are two cases: 1) If no counter-sharing happens in the d_l hashed counters, then these counters will keep the same values all the time and overflow concurrently. 2) If counter-sharing happens in one or more counters of the d_l hashed counters, then the values of these hashed counters may differ from each other. The method of incrementing the smallest counter(s) will always tend to narrow the differences of values. Until the d_l hashed counters all reach the value of $2^{\delta_l}-1$, can the d_l hashed counters have the opportunity to overflow. Therefore, these d_l hashed counters will always overflow concurrently. Therefore, we can just use the flow ID of e as the key of the d_h hash functions $h_h^i(.)(1 \leqslant i \leqslant d_h)$ to locate the d_h hashed counters $L_h[h_h^1(e)]$, $L_h[h_h^2(e)]$, ..., $L_h[h_h^{d_h}(e)]$ at the high layer. Then we increment the smallest d_h hashed counter(s) at the high layer and set all the d_l hashed counters at the low layer to zero. In this way, the high layer records the number of overflows of this flow.

Query: When querying a flow e, we first compute the d_l hash functions $h_l^1(e), h_l^2(e), ..., h_l^{d_l}(e)$ $(1 \leqslant h_l(.) \leqslant w_l)$ and locate the d_l hashed counters $L_l[h_l^1(e)]$, $L_l[h_l^2(e)]$, ..., $L_l[h_l^{d_l}(e)]$.Then we get the value of the smallest counter(s) among the d_l hashed counters at the low layer and we use V_l to represent it. Similarly, we can use the flow ID of e as the key to compute d_h hash functions and acquire the value of the smallest counter(s) among the d_h hashed counters at the high layer. We represent it as V_h. Then we report $V_l + V_h \times 2^{\delta_l}$ as the estimated size of flow e.

As mentioned in section I-A, the flow size distribution in the real network is highly skewed. The elephant flows are often considered as more important and the counter size should be specially tailored for them in the CM [1], CU [17] and Count [18] sketch. In this case, the unified counters of conventional sketches are too large for the mice flows to "fill up". As a result, the waste of high-order bits in most counters leads to poor memory efficiency and poor accuracy as well. Fortunately, the two-layer OM sketch with the hierarchical counter-sharing technique can just address this issue. We use the low layer to efficiently record the information of the mice flows, so as to alleviate the memory waste, and use the high layer to only record the information of the elephant flows, so as to improve the accuracy. Therefore, δ_l is set to a small value (*e.g.*, 4 bits) to only accommodate the sizes of mice flows. And δ_h needs to be set to a relatively large value (*e.g.*, 16 bits) to accommodate the sizes of elephant flows.

2) Word Acceleration:

Word Constraint: In the word constraint technique, we set the counter size δ_l at the low layer to 4 bits and confine the d_l hashed counters at the low layer within one machine word. We call this machine word the **hashed word** of this flow. Let W be the number of bits in a machine word. Besides, the low layer L_l is associated with $d_l + 1$ independent hash functions. The first hash function is used to locate one machine word at the low layer and the other d_l hash functions are used to locate the d_l counters in this machine word. For the high layer, we constrain the d_h hashed counters within several (*e.g.*, 2) machine words and scatter these counters over these machine words evenly, since δ_h (*e.g.*, 16 bits) should be larger than δ_l (*e.g.*, 4 bits). Specifically, the high layer L_h is associated with d_h+2 independent hash functions. The first two hash functions locate two machine words at the high layer respectively, and the other d_h hash functions locate the d_h counters within the two machine words. We represent the i^{th} machine word at the low layer as $L_l^w[i]$ and the j^{th} counter in the word $L_l^w[i]$ as $L_l^w[i][j]$. Similarly, we have $L_h^w[i]$ and $L_h^w[i][j]$.

In modern CPUs, a machine word is usually 64 bits wide. In the GPU architecture, the size of a machine word is much larger, as the GPU can read/write 1024 bits in one memory access [25]. Therefore, a machine word can contain enough counters to be hashed by d_l or $d_h/2$ hash functions. In addition, as the flow size distribution in the real network is highly skewed [15], [16], the probability of accessing the high layer for each insertion is very small (*e.g.*, $1/20$). Therefore, the average number of memory accesses for each insertion is close to 1 (*e.g.*, $1 + 1/20 * 2 = 1.1$).

Hash Bit: Furthermore, we leverage the hash bit technique from the literature [19] to reduce the number of hash computations. We split one 64-bit (64 bits are enough in practical per-flow measurement tasks) hash value into several bit arrays to locate one or several machine words and d_l or d_h offsets in the corresponding machine words. In other words, we use one 64-bit hash function to serve as the d_l+1 hash functions at the low layer or the $d_h + 2$ hash functions at the high layer. Besides, suppose the probability of accessing the high layer is around $1/20$. In this case, the average number of hash computations for each insertion is close to 1 ($1 + 1/20 * 1 = 1.05$).

3) Fingerprint Check:

Only based on the two techniques above, our OM sketch cannot achieve satisfying accuracy and query speed. The reason is that mice flows always need to query the high layer and are prone to getting a non-zero V_h which accounts for a large portion of the final estimated result. As a result, the estimated sizes of mice flows are often largely overestimated and the query speed is also strictly limited. To address this issue, we propose the fingerprint check technique.

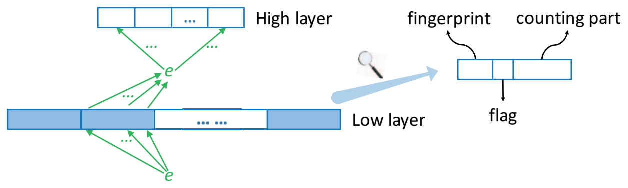

Fig. 2: Final structure of our OM sketch.

Fingerprint and flag: As shown in Figure 2, each word at the low layer is split into three parts: *ρ-bit fingerprint, 1-bit flag, counting part*. All the three parts are set to zero initially. The d_l hashed counters within one word at the low layer will be chosen from the counting part which consists of λ δ_l-bit counters. The parameters need to satisfy $W = \rho + 1 + \lambda \times \delta_l$. Note that in the following paper, w_l *only represents the number of counters in the counting parts of all words at the low layer*. For convenience, we use $L_l^w[i].fp, L_l^w[i].flag, L_l^w[i].count$ to represent the three parts of the word $L_l^w[i]$. For an arbitrary flow e, let $o_l^i(e)(1 \leqslant i \leqslant d_l)$ denote the offsets of its d_l hashed counters in its corresponding counting part at the low layer. Similarly, we have $o_h^i(e)(1 \leqslant i \leqslant d_h)$. The fingerprint of this flow $fp(e)$ (ρ bits) is derived from the d_l offsets through several logical operations. Note that we call $fp(e) \nsubseteq L_l^w[i].fp$ if $(L_l^w[i].fp \ \& \ fp(e)) \neq L_l^w[i].fp$.

Insertion: The procedure for insertions is shown in Algorithm 1. If overflows are not to happen at the low layer, we just increment the smallest hashed counter(s). If overflows happen, we first set all the d_l hashed counters to zero, and then check whether this flow is the first overflowed flow in this hashed word. If it is not, we set the $L_l^w[i].flag$ to one. Therefore, the $L_l^w[i].flag$ tells whether the number of overflowed flows hashed in the word $L_l^w[i]$ exceeds one. Line 7 records $fp(e)$ in the $L_l^w[i].fp$ part by the logical OR operation. We see that $L_l^w[i].fp$ is the result of logical OR of fingerprints of all the overflowed flows hashed in the word $L_l^w[i]$. Line 8 increments the smallest hashed counter(s) at the high layer.

Algorithm 1: Insertion algorithm of the OM sketch

```
Input: flow $e$, the hashed words $L_l^w[i]$, the offsets
       $o_l^i(e)(1 \leqslant i \leqslant d_l)$, and the fingerprint $fp(e)$.
Output: update the OM sketch.
1 if $L_l^w[i][o_l^k(e)](1 \leqslant k \leqslant d_l)$ are not to overflow then
2     increment the smallest hashed counter(s) at layer $L_l$;
3 else
4     set $L_l^w[i][o_l^k(e)](1 \leqslant k \leqslant d_l)$ to 0;
5     if $L_l^w[i].fp \neq 0 \wedge fp(e) \nsubseteq L_l^w[i].fp$ then
6         $L_l^w[i].flag \leftarrow 1$;
7     $L_l^w[i].fp \leftarrow L_l^w[i].fp \mid fp(e)$;
8     increment the smallest hashed counter(s) at layer $L_h$;
```

Query: The procedure for queries is shown in Algorithm 2. We first get the value of the smallest hashed counter(s) at the low layer. Then we check if $L_l^w[i].flag$ is one. 1) If it is, the number of overflowed flows hashed in this word must exceed one. Thus we need to further check whether $fp(e) \nsubseteq L_l^w[i].fp$. If $fp(e) \nsubseteq L_l^w[i].fp$, then this flow did not overflow. We just return V_l.2) If it is not, the number of overflowed flows hashed in this word must be one or zero. We need to further check whether $fp(e) \neq L_l^w[i].fp$. If $fp(e) \neq L_l^w[i].fp$, then this flow did not overflow. We just return V_l.3) In the rest of the cases, this flow overflowed in a high probability. We need to query the high layer and get the V_h. Then we return $V_l + V_h \times 2^{\delta_l}$ as the estimated flow size.

Algorithm 2: Query algorithm of the OM sketch

```
Input: flow $e$, the hashed words $L_l^w[i]$, the offsets
       $o_l^i(e)(1 \leqslant i \leqslant d_l)$, and the fingerprint $fp(e)$.
Output: the esitmated size of flow $e$.
1 $V_l \leftarrow$ value of the smallest hashed counter(s) at layer $L_l$;
2 if $L_l^w[i].flag = 1$ then
3     if $fp(e) \nsubseteq L_l^w[i].fp$ then
4         return $V_l$;
5 else
6     if $fp(e) \neq L_l^w[i].fp$ then
7         return $V_l$;
8 $V_h \leftarrow$ value of the smallest hashed counter(s) at layer $L_h$;
9 return $V_l + V_h \times 2^{\delta_l}$;
```

By checking the fingerprint parts at the low layer, we can effectively prevent mice flows from querying the high layer so as to improve the accuracy and query speed. And we can make sure that it never prevents elephant flows from querying the high layer. In this way, the probability of accessing the high layer for each query is reduced to a low level (*e.g.*, $1/20$). Therefore, for each query, the average number of memory accesses is close to 1 (*e.g.*, $1 + 1/20 * 2 = 1.1$), and the average number of hash computations is analogous (*e.g.*, $1 + 1/20 * 1 = 1.05$).

IV. Performance Evaluation

A. Metrics

Average Absolute Error (AAE): AAE is defined as $\frac{1}{|N|}\sum_{i=1}^{N}|f_i - \widehat{f_i}|$ where N is the total number of distinct flows, f_i is the real size of the i^{th} flow and $\widehat{f_i}$ is the estimated size of this flow.

Average Relative Error (ARE): ARE is defined as $\frac{1}{|N|}\sum_{i=1}^{N}|f_i - \widehat{f_i}|/f_i$.

Correct Rate ($\mathcal{C}_r$): $\mathcal{C}_r$ is defined as N_{acc}/N where N_{acc} is the number of distinct flows whose estimated size equals to its real size.

The Average Number of Memory Accesses: This metric can reflect the performance of the sketches implemented on hardware (*e.g.*, FPGA, ASIC).

Throughput: We simulate how sketches actually insert and query on CPU platform, and calculate the throughput of insertions and queries. We repeat each experiment 100 times to minimize the accidental derivation.

B. Experimental Setup

We use 10 IP traces coming from the main gateway of our campus. The number of packets of each trace is 10M and the number of distinct flows is approximately 1M.

We implement the sketches with C++. In terms of hash functions, we use 64-bit Bob hash function [26] for the OM sketch while other sketches use 32-bit Bob hash function. In the OM sketch, we set $d_l = d_h = 4, \delta_l = 4, \delta_h = 16, \rho = 7$, and we constrain the hashed counters with two machine words at the high layer. In other sketches, the counter size is set to 16 bits. The size of one machine word is 64 bits. The memory size of each sketch is 1MB unless noted otherwise. Besides, for the OM sketch, we allocate $\frac{1}{3}$ memory for the high layer and $\frac{2}{3}$ for the low layer, which helps achieve better performance.

We performed all the experiments on a machine with 12-core CPUs (24 threads, Intel Xeon CPU E5-2620 @2 GHz) and 62GB total DRAM memory running Ubuntu 14.04.

C. Accuracy

Our experimental results show that the AAEs of the CM, CU, C sketch are 3.47, 1.80, 2.15 times higher than the AAE of the OM sketch, respectively. Figure 3 plots the AAEs of different sketches on different IP traces.

Our experimental results show that on different memory sizes, the AAEs of the CM, CU, C sketch are 4.49, 2.12, 2.26 times higher than the AAE of the OM sketch, respectively. Figure 4 plots the AAEs of different sketches on different memory sizes increasing from 0.5MB to 1.5MB with a step of 0.1MB.

Our experimental results show that the AREs of the CM, CU, C sketch are 6.99, 4.11, 4.35 times higher than the ARE of the OM sketch, respectively. Figure 5 plots the AREs of different sketches on different IP traces.

Our experimental results show that on different memory sizes, the AREs of the CM, CU, C sketch are 10.64, 5.98, 5.27 times higher than the ARE of the OM sketch, respectively. Figure 6 plots the AREs of different sketches on different memory sizes increasing from 0.5MB to 1.5MB with a step of 0.1MB.

Our experimental results show that on different memory sizes, the C_r of the OM sketch is 16.50, 4.71, 8.25 times higher than the C_rs of the CM, CU, C sketch, respectively. Figure 7 plots the C_rs of different sketches on different memory sizes increasing from 0.5MB to 1.5MB with a step of 0.1MB.

D. Speed

Our experiment results show that the CM, CU, C and OM sketch need about 4, 4, 4, 1.08 memory accesses for each insertion, respectively. Figure 8 plots the average numbers of memory accesses for each insertion on different IP traces.

Our experiment results show that the CM, CU, C and OM sketch need about 4, 4, 4, 1.09 memory accesses for each query, respectively. Figure 9 plots the average numbers of memory accesses for each query on different IP traces.

Our experiment results show that the insertion throughput of the OM sketch is 2.50, 2.62, 4.20 times higher than those of the CM, CU, C sketch, respectively. Figure 10 plots the insertion throughput of different sketches on different IP traces. We observe that the word constraint technique accelerates the process of insertions significantly.

Our experiment results show that the query throughput of the OM sketch is 2.18, 2.15, 5.20 times higher than those of the CM, CU, C sketch, respectively. Figure 11 plots the query throughput of different sketches on different IP traces.

V. Conclusion

Sketch is one kind of useful data structure especially in the field of per-flow measurement. In this paper, we propose a novel sketch - the OM sketch, which achieves close to one memory access and one hash computation for each insertion or query while achieving high accuracy. Our OM sketch consists of three key techniques: hierarchical counter-sharing, word acceleration and fingerprint check. Experimental results show that our OM sketch can achieve a much better performance than the state-of-the-art in terms of both speed and accuracy. We believe that our OM sketch can be well applied in the field of per-flow measurement. We have released all the relevant source code at Github [2].

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their thoughtful suggestions. This work is partially supported by Primary Research & Development Plan of China (2016YFB1000304), NSFC (61232004, 61672061), NSF (STC-1562485), SHENZHEN Research Project JCYJ20160330095313861, the Open Project Funding of CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences, and Special Fund for strategic pilot technology Chinese Academy of Sciences (XDA06010302).

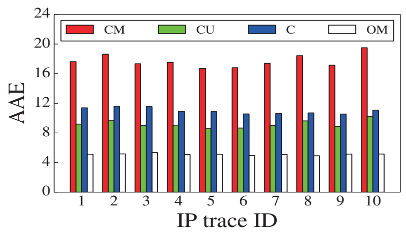

Fig. 3: AAE vs. IP traces.

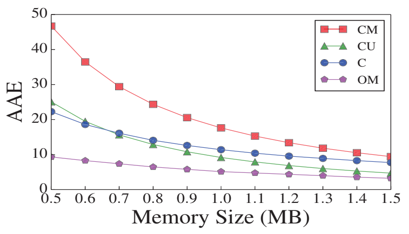

Fig. 4: AAE vs. memory sizes.

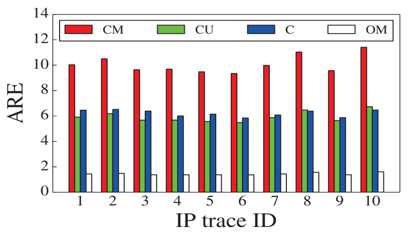

Fig. 5: ARE vs. IP traces.

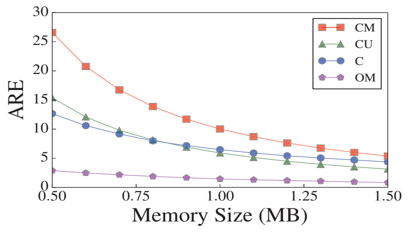

Fig. 6: ARE vs. memory sizes.

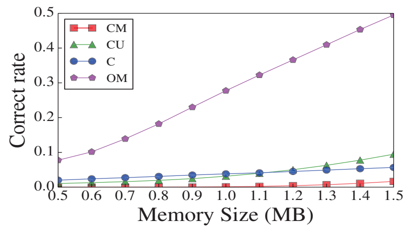

Fig. 7: $\mathcal{C}_r$ vs. memory sizes.

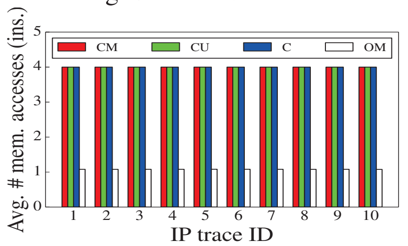

Fig. 8: Average # memory accesses for each insertion vs. IP traces.

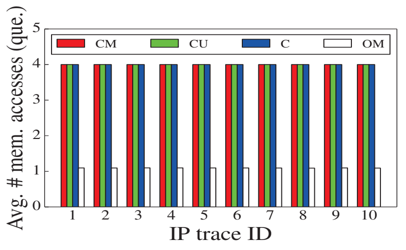

Fig. 9: Average # memory accesses for each query vs. IP traces.

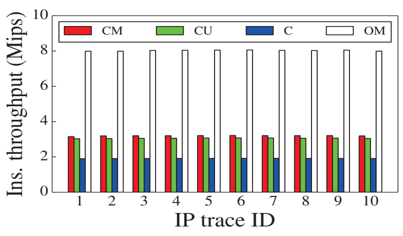

Fig. 10: Insertion throughput vs. IP traces.

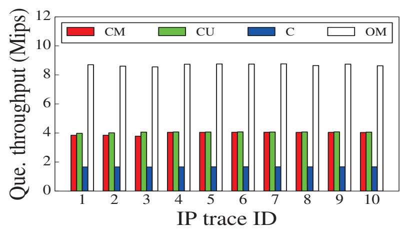

Fig. 11: Query throughput vs. IP traces.

References

[1] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[2] "Source code of the OM sketch and related sketches." https://github.com/zhouyangpkuer/OMsketch.

[3] N. Duffield, C. Lund, and M. Thorup, "Learn more, sample less: control of volume and variance in network measurement," *IEEE Transactions on Information Theory*, vol. 51, no. 5, pp. 1756–1775, 2005.

[4] Y. Zhou, Y. Zhou, S. Chen, and O. P. Kreidl, "Limiting self-propagating malware based on connection failure behavior," in *Proc. of Seventh International Conference on Network and Communications Security (NCS)*, 2015.

[5] D. Plonka, "Flowscan: A network traffic flow reporting and visualization tool." in *LISA*, 2000, pp. 305–317.

[6] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: an efficient algorithm for finding heavy hitters," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, pp. 5–5, 2008.

[7] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. ACM, 2003, pp. 234–247.

[8] 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*. ACM, 2004, pp. 207–212.

[9] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2327–2339, 2006.

[10] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: a novel counter architecture for per-flow measurement," *Proc. ACM SIGMETRICS*, vol. 36, no. 1, pp. 121–132, 2008.

[11] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1622–1634, 2012.

[12] G. Cormode and M. Garofalakis, "Sketching streams through the net: Distributed approximate query tracking," in *Proc. VLDB*, 2005.

[13] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1530–1541, 2008.

[14] D. Thomas, R. Bordawekar, and et al., "On efficient query processing of stream counts on the cell processor," in *Proc. IEEE ICDE*, 2009.

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

[16] K. Cheng, L. Xiang, M. Iwaihara, H. Xu, and M. M. Mohania, "Time-decaying bloom filters for data streams with skewed distributions," in *International Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications, 2005. Ride-Sdma*, 2005, pp. 63–69.

[17] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *ACM SIGMCOMM CCR*, vol. 32, no. 4, 2002.

[18] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Automata, Languages and Programming*. Springer, 2002.

[19] Y. Qiao, T. Li, and S. Chen, "One memory access bloom filters and their generalization," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1745–1753.

[20] Y. Lu and B. Prabhakar, "Robust counting via counter braids: An error-resilient network measurement architecture," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 522–530.

[21] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[22] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," in *Proc. ACM SIGCOMM*, 1998.

[23] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proc. ACM SIGMOD*, 2003.

[24] J. Aguilar-Saborit, P. Trancoso, V. Muntes-Mulero, and J.-L. Larriba-Pey, "Dynamic count filters," *ACM SIGMOD Record*, pp. 26–32, 2006.

[25] "Cuda toolkit documentation." http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#coalesced-access-to-global-memory.

[26] "Hash website," http://burtleburtle.net/bob/hash/evahash.html.