Accurate Per-Flow Measurement with Bloom Sketch

Yang Zhou, Hao Jin, Peng Liu, Haowei Zhang, Tong Yang, Xiaoming Li
Peking University, {zhou.yang, jin.hao, liu.peng, zhanghw.alpq, yang.tong, lxm}@pku.edu.cn

***Abstract*—Sketch is a probabilistic data structure, and is widely used for per-flow measurement in network. The most common sketches are the CM sketch and its several variants. However, given a limited memory size, these sketches always significantly overestimate some flows, exhibiting poor accuracy. To address this issue, we proposed a novel sketch named the Bloom sketch, combining the sketch with the Bloom filter, another well-known probabilistic data structure widely used for membership queries. Extensive experiments based on real IP traces show that our Bloom sketch achieves up to $14.47\times$ higher accuracy compared with the CM sketch, while exhibiting comparable insertion and query speed. Our source code is available at Github [1].**

I. Introduction

Per-flow measurement, which refers to estimating the *number of packets in each flow* (*i.e.*, flow size), is a fundamental issue in modern network. It can help detect SYN flooding attack [2], alleviate network congestion [3], find heavy hitters [4] and heavy changes [5] in intrusion detection systems, *etc.* Due to the high speed and huge volume of network traffic, one kind of probabilistic data structures called *sketches* is extensively used for per-flow measurement. The sketch targets at performing approximate per-flow measurement at line speed with a limited memory size. There are three typical sketches: the CM [6], CU [7], and Count sketch [8]. Each of these three sketches is a counter array with several hash functions. When processing a packet, they extract the flow ID (*e.g.*, source IP, destination IP, 5-tuple, *etc.*) from this packet, and locate multiple counters in their counter array by calculating multiple hashes on the flow ID. Then, CM increments all the hashed counters by one; CU only increments all the smallest counter(s). When given a flow ID, the CM/CU sketch reports the minimum value of the hashed counters as the corresponding flow size. For the count sketch and more, we refer readers to the literature [8]–[13].

However, all conventional sketches cannot work well in real network traffic. In real network traffic, the distribution of flow sizes is skewed: a few flows (*i.e.*, elephant flows) have large sizes while most flows (*i.e.*, mice flows) only have small sizes [14]. As conventional sketches use multiple arrays of fixed-size counters to perform the counting, the counter size should be determined by the *most elephant flows* in network traffic. Therefore, numerous mice flows cannot *"fill up"* the counters that they are hashed to. That is, the high-order bits of most counters are wasted, which leads to poor memory efficiency. Besides, given a limited memory size, conventional sketches suffer from another problem: some mice flows are apt to be significantly overestimated due to severe hash collisions with elephant flows. We call this problem the *elevation effects* of elephant flows. In this paper, we proposed the Bloom sketch to *perform memory-efficient counting based on hierarchical counter arrays and reduce the elevation effects by employing multiple Bloom filters.*

Corresponding author: Tong Yang. Email: {yangtongemail@gmail.com}.

This work is supported by Primary Research & Development Plan of China (2016YFB1000304), National Basic Research Program of China (973 Program, 2014CB340405), NSFC (61672061), the OpenProject Funding of CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology, Chinese Academy of Sciences.

II. The Bloom Sketch Algorithm

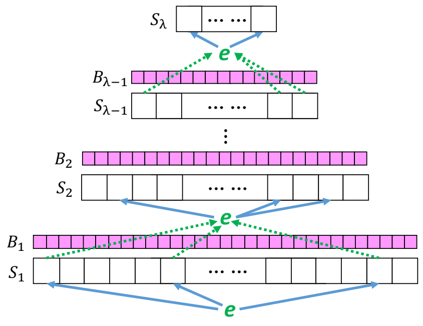


Fig. 1: Hierarchical counter arrays + Bloom filter layer.

Data structure: As shown in Figure 1, the Bloom sketch consists of λ sketch layers (*i.e.*, counter arrays): $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_\lambda$, with small counters, and $\lambda - 1$ Bloom filter layers (*i.e.*, bit arrays): $\mathcal{B}_1, \mathcal{B}_2, ..., \mathcal{B}_{\lambda-1}$.

Insertions: When processing a packet with flow ID e, we first locate d_1 hashed counters at $\mathcal{S}_1$ by calculating d_1 hashes, and increment the smallest hashed counter(s). If one or more hashed counters overflow at $\mathcal{S}_1$, we have the following guarantee: e *must cause their* d_1 *hashed counters to overflow concurrently*, since we always increment the smallest counter(s). This guarantee provides us an opportunity to isolate each sketch or Bloom filter layer from the others to a certain extent. We can just use e as the hash key of the d_2 hash functions to locate d_2 hashed counters at $\mathcal{S}_2$. Then we increment the smallest hashed counter(s) at $\mathcal{S}_2$ and set the d_1 hashed counters at $\mathcal{S}_1$ to 0. In addition, we uses e as the hash key to locate d'_1 hashed bits at $\mathcal{B}_1$ and set these bits to one. Similarly, the property of concurrent overflowing also applies to $\mathcal{S}_2$ and more. Therefore, if $\mathcal{S}_2$ overflows also, we can similarly resort to $\mathcal{S}_3$ to record the number of overflows, and set the d'_2 hashed bits at $\mathcal{B}_2$ to one. Such operations will continue recursively, until no overflow happens at one certain sketch layer.

Query: When given a flow ID e, we first get the smallest value V_1 of the d_1 hashed counters at $\mathcal{S}_1$. Then we need to identify whether e has reached the higher layers during insertions by checking whether the d'_1 hashed bits at the layer $\mathcal{B}_1$ are all one. 1) If the d'_1 hashed bits are not all one, we just report the V_1 as the estimated flow size of e. 2) Otherwise, we need to do the same operation at $\mathcal{S}_2$ and $\mathcal{B}_2$. Such operations will be performed continuously until the $d'_{\lambda'}$ hashed bits at $\mathcal{B}_{\lambda'}$ are not all one. Then we report the $\sum_{i=1}^{\lambda'}\left(V_i \times 2^{\sum_{j=1}^{i-1}\delta_j}\right)$ as the estimated flow size of e. Due to the isolation among the sketch or Bloom filter layers, querying each of these layers can be performed in parallel.

One Memory Access Bloom Filter: To accelerate the operations at the Bloom filter layers, we employ the one memory access Bloom filter [15]. Specifically, we confine the multiple hashed bits at the Bloom filter layer within one machine word. Besides, we split one 64-bit hash value into several bit strings with user-defined length. We use these bit strings to first locate a machine word at one certain Bloom filter layer, and then locate multiple hashed bits within this machine word. In this way, the cost that each insertion or query spent at each Bloom filter layer is reduced to one memory access and one hash computation.

Analysis: In the Bloom sketch, each sketch layer functions like an independent sketch to record a small part of each flow size. Due to the skewed distribution of flow sizes, the higher sketch layer accessed by fewer packets can use less memory to achieve the same accuracy. By leveraging a suitable scheme of memory allocation, the memory efficiency can be guaranteed because almost all bits of each counter are fully utilized. The Bloom filter layers can reduce the elevation effects by helping check the overflowing status at the corresponding sketch layers.

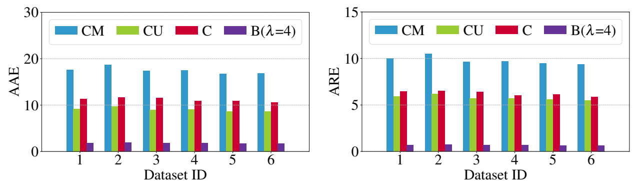
III. Performance Evaluation

Experiment Setup: We use 6 real IP traces from the main gateway of our campus, and use 5-tuple as the flow ID. We set $\lambda = 4$, locate 4 hashed counters or bits at each layer, and allocate $0.1/0.9$MB memory to Bloom filter/sketch layers. The memory allocation within each type of layers follows geometric series with common ratio of 0.125. We make source code available at Github [1].

Average Absolute Error (AAE): AAE is defined as $\frac{1}{|N|}\sum_{i=1}^{N}|f_i - \widehat{f_i}|$, where N is the number of flows and $f_i/\widehat{f_i}$ is the real/estimated size of flow e_i. Figure 2(a) plots the AAEs of different sketches on different IP tracecs. *Our experimental results show that averagely, the AAE of the Bloom sketch is 9.82, 5.10, and 6.26 times lower than the AAEs of the CM, CU and C sketch, respectively.*

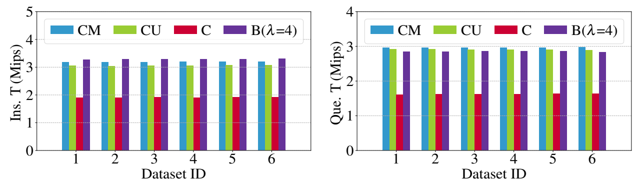
Average Relative Error (ARE): ARE is defined as $\frac{1}{|N|}\sum_{i=1}^{N}(|f_i - \widehat{f_i}|/f_i)$. Figure 2(b) plots the AREs of different sketches on different IP tracecs. *Our experimental results show that averagely, the ARE of the Bloom sketch is 14.47, 8.52, and 9.19 times lower than the AREs of the CM, CU and C sketch, respectively.*

Throughput (T): We calculate the throughput for each insertion or query with unit of mega-instructions per second (Mips).



(a) AAE vs. Dataset ID. (b) ARE vs. Dataset ID.

Fig. 2: Accuracy comparison on different IP traces.



(a) Ins. throughput vs. Dataset ID. (b) Que. throughput vs. Dataset ID.

Fig. 3: Throughput comparison on different IP traces.

Figure 3(a) and 3(b) plot the insertion and query throughputs of different sketches on different IP traces, respectively.

References

[1] "Source code of the Bloom sketch and related sketches," https://github.com/zhouyangpkuer/BloomSketch.

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

[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] 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.

[5] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proc. IMC*. ACM, 2003, pp. 234–247.

[6] 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.

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

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

[9] T. Yang, L. Liu, Y. Yan, M. Shahzad, Y. Shen, X. Li, B. Cui, and G. Xie, "Sf-sketch: A fast, accurate, and memory efficient data structure to store frequencies of data items," in *Proc. IEEE ICDE*, 2017.

[10] Y. Zhou, P. Liu, H. Jin, T. Yang, S. Dang, and X. Li, "One memory access sketh: a more accurate and faster sketch for per-flow measurement." IEEE Globecom, 2017.

[11] J. Gong, T. Yang, Y. Zhou, D. Yang, S. Chen, B. Cui, and X. Li, "Abc: a practicable sketch framework for non-uniform multisets." IEEE Bigdata, 2017.

[12] L. Wang, Z. Cai, H. Wang, J. Jiang, T. Yang, B. Cui, and X. Li, "Fine-grained probability counting: Refined loglog algorithm." IEEE Bigcomp, 2018.

[13] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li, "Pyramid sketch: a sketch framework for frequency estimation of data streams," *Proc. VLDB*, vol. 10, no. 11, pp. 1442–1453, 2017.

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

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