Diamond Sketch: Accurate Per-flow Measurement for Real IP Streams

Tong Yang*, Siang Gao*, Zhouyi Sun*, Yufei Wang*, Yulong Shen†, Xiaoming Li*
*Department of Computer Science, Peking University, China †Xidian University, China

***Abstract*—Existing sketches often have low memory efficiencies when performing per-flow measurement tasks on skewed IP streams. In this paper, we propose Diamond Sketch, a novel sketch that dynamically assigns an appropriate number of atom sketches to each flow on demand, improving the accuracy considerably while keeping a comparable speed.**

I. Introduction

Per-flow measurement is a critical issue in computer networks. It provides information for anomaly detection, capacity planning, accounting and billing, and service provision [1], [2]. One fundamental problem in per-flow measurement is to estimate the flow size, which is the number of packets in each flow. A flow is often identified by a certain combination of fields in the five-tuple in the packet's header: source IP address, destination IP address, source port number, destination port number, and protocol type. Real network flows have two characteristics: *high-speed* and *non-uniform distribution*. On the one hand, the speed of network traffic is so high that it is very hard to make a precise record of sizes of flows [3]. On the other hand, the sizes of network flows are usually non-uniformly distributed [4]. A small part of flows have very large sizes (called *elephant flows*) while most flows have small sizes (called *mice flows*). A typical distribution of this kind of network traffic is Zipfian [5], and network traffic following Zipfian distribution are called *skewed* traffic.

Due to the high speed of network traffic, approximately recording and estimating data with sketches has gained popularity [4], [6]–[9]. Sketches are probabilistic data structures, and can achieve small memory footprints, high accuracy, and fast speeds of insertion and query. Unfortunately, most existing sketches cannot work well on skewed network traffic. Conventional sketches (such as CM [10], CU [11], Count [12], CSM [13] sketches) use counters that are of the same size to store the number of packets. Elephant flows are often considered to be more important than mice flows, thus the counters need to be large enough to represent the largest size of the elephant flows. However, the large quantity of mice flows means that most counters will just represent a small value, and the higher bits in these counters are all 0, leading to a waste of memory.

Co-primary Authors: Tong Yang and Siang Gao.

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.

In this paper, we propose a new sketch, namely the Diamond sketch, as it takes on a diamond shape. The *key idea* of Diamond is to assign atom sketches to flows on demand. Specifically, an appropriate number of atom sketches are assigned dynamically and automatically to record the sizes of elephant and mice flows, thus optimizing memory efficiency. For mice flows, we use only one or two atom sketches to record their sizes. For elephant flows, we dynamically increase the number of atom sketches to record their sizes.

II. The Diamond Sketch

A. Data Structure

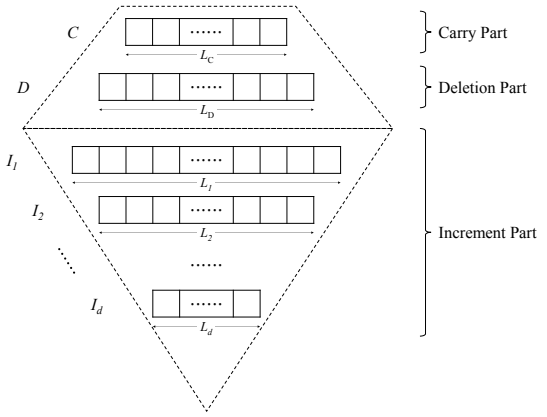


Fig. 1. Data structure of the Diamond Sketch.

As shown in Figure 1, a Diamond Sketch consists of three parts: increment part, carry part, and deletion part. **Increment part** together with carry part records the size of each flow that has been inserted. It consists of d atom sketches, where we denote the i^{th} atom sketch with I_i. Each atom sketch I_i is composed of L_i counters, and each counter contains w_1 bits. We denote the j^{th} counter of the i^{th} atom sketch with $I_i[j]$. $\{L_1, L_2, ..., L_d\}$ is a decreasing sequence of numbers. **Carry part** records the *overflow depth*[1] of each flow. It is an atom sketch composed of L_{C} counters, each of which contains w_2 bits ($2^{w_2} \geqslant d$ is a requisite). We denote this atom sketch with $\mathcal{C}$, and the j^{th} counter with $C[j]$. **Deletion part** is used to support deletions. It is an atom sketch composed of L_{D} counters, each of which contains w_3 bits. We denote this atom sketch with $\mathcal{D}$, and the j^{th} counter with $D[j]$.

[1]Suppose a flow causes overflows in atom sketch I_1 and I_2 when being inserted, and is successfully inserted into I_3 without overflows, then the deepest sketch this flow has reached is I_3, and 3 is called the *overflow depth*.

Each atom sketch I_i, $\mathcal{C}$, and $\mathcal{D}$ is associated with k_1, k_2, and k_3 hash functions, whose output is uniformly distributed in the range $[1, L_i]$, $[1, L_C]$, and $[1, L_D]$. And we denote the j^{th} hash function with $h_j^i(.)$, $h_j^{\mathrm{C}}(.)$, and $h_j^{\mathrm{D}}(.)$, respectively.

B. Operations

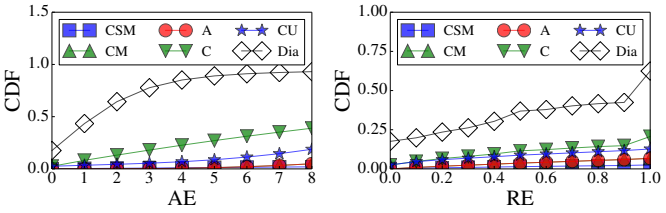
The **insertion** of a packet with flow ID e requires update in first the increment part and second the carry part. We compute the k_1 hash functions of the first atom sketch I_1 and increment the smallest one(s) of the k_1 counters $I_1[h_1^1(e)], I_1[h_2^1(e)], ..., I_1[h_{k_1}^1(e)]$ (we call them k_1 *mapped counters* for short) by 1. If more than one counters have the smallest value, we increment them all by 1. If all the k_1 counters hold the value $2^{w_1} - 1$, which is the largest number that w_1 bits can represent, then incrementing them by 1 will result in *overflows*. When the k_1 mapped counters in I_i overflow, the following steps need to be taken: **1)** Set the values of all k_1 mapped counters to 0; **2)** Increment the smallest one(s) of the k_1 mapped counters of I_{i+1} by 1; **3)** If all the k_1 mapped counters in I_{i+1} overflow, repeat step 1 and 2 until the *termination condition* is satisfied, and we record the overflow depth *dep*. There are two termination conditions: there is no overflow in I_i $(1 \leqslant i \leqslant d)$ (we set *dep* to i) or the last atom sketch I_d overflows (we set *dep* to d); **4)** For each of the k_2 mapped counters in $\mathcal{C}$, if its value is less than $dep - 1$, we set its value to $dep - 1$; otherwise, we do nothing.

The **deletion** operation is not requisite in per-flow measurement, but can be added if needed in other scenarios. To delete a packet e, the Diamond Sketch only updates its deletion part. We check the k_3 mapped counters of $\mathcal{D}$, and if all the k_3 counters are $2^{w_3} - 1$, which is the biggest number that w_3 bits can represent, we do nothing; otherwise, we increment the smallest one(s) among the k_3 counters by 1.

To **query** the size of a flow with ID e, we need to check the status of all the three parts. **1)** We check the carry part. Specifically, we compute the k_2 hash functions of $\mathcal{C}$ and get the smallest value v_C among $C[h_1^{\mathrm{C}}(e)], C[h_2^{\mathrm{C}}(e)], ..., C[h_{k_2}^{\mathrm{C}}(e)]$. **2)** We check the increment part. Let $dep = 1 + v_C$. We need to check *dep* atom sketches $I_1, I_2, ..., I_{dep}$. For each atom sketch I_i, we compute k_1 hash functions and get the smallest value V_i of those k_1 mapped counters $I_i[h_1^i(e)], I_i[h_2^i(e)], ..., I_i[h_{k_1}^i(e)]$. Once all *dep* smallest values have been calculated, the query result determined by increment part and carry part is computed by the following formula: $V_{insert} = \sum_{i=1}^{dep} V_i \cdot (2^{w_1})^{i-1}$. **3)** We check the deletion part. Specifically, we compute the k_3 hash functions of $\mathcal{D}$ and return the smallest value (V_{delete}) among $D[h_1^{\mathrm{D}}(e)], D[h_2^{\mathrm{D}}(e)], ..., D[h_{k_3}^{\mathrm{D}}(e)]$. **4)** We compute $V_{insert} - V_{delete}$ as the query result.

III. Evaluation

We use real IP trace streams as experimental datasets, which are captured by the main gateway of our campus. We compare Diamond Sketch with CU [11], CM [10], Augmented [14], Count [12], and CSM [13] sketches. The metrics used are absolute error (AE) and relative error (RE).



(a) Absolute Error CDF. (b) Relative Error CDF.

Fig. 2. AE-CDF and RE-CDF on IP streams. *Dia* stands for *Diamond sketch*.

As shown in Fig. 2, the percentage of flows whose AE is no bigger than 1 of our Diamond sketch is 43.39%, which is 124.23, 2018.3, 1823.25, 5.3, and 12.5 times higher than the corresponding percentages of CSM, CM, A, C, CU sketches, respectively; the percentage of flows whose RE is less than 1.0 of our Diamond sketch is 62.46%, which is 24.6, 9.29, 9.29, 3.02, and 4.91 times higher than the corresponding percentages of CSM, CM, A, C, CU sketches, respectively. The results show that the performance of Diamond is far better than other sketches in terms of accuracy.

References

[1] 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.

[2] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a spread estimator in small memory," in *Proc. IEEE INFOCOM 2009*.

[3] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: a better netflow for data centers," in *Proc. USENIX NSDI*, 2016.

[4] G. Cormode, "Sketch techniques for approximate query processing," *Synposes for Approximate Query Processing: Samples, Histograms, Wavelets and Sketches, Foundations and Trends in Databases. NOW publishers*, 2011.

[5] D. M. Powers, "Applications and explanations of zipf's law," in *Proceedings of the joint conferences on new methods in language processing and computational natural language learning*. Association for Computational Linguistics, 1998, pp. 151–160.

[6] T. Yang, A. X. Liu, M. Shahzad, D. Yang, Q. Fu, G. Xie, and X. Li, "A shifting framework for set queries," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 3116–3131, 2017.

[7] T. Yang, A. X. Liu, M. Shahzad, Y. Zhong, Q. Fu, Z. Li, G. Xie, and X. Li, "A shifting bloom filter framework for set queries," *Proceedings of the VLDB Endowment*, vol. 9, no. 5, pp. 408–419, 2016.

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

[9] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig, "Cold filter: A meta-framework for faster and more accurate stream processing," in *Proc. SIGMOD*, 2018.

[10] 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.

[11] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems (TOCS)*, vol. 21, no. 3, pp. 270–313, 2003.

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

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

[14] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proc. ACM SIGMOD*, 2016, pp. 1449–1463.