Bubble Sketch: A High-performance and Memory-efficient Sketch for Finding Top-k Items in Data Streams

Lu Cao
Harbin Institute of Technology
Shenzhen, China

Qilong Shi
Tsinghua University
Beijing, China

Yuxi Liu
Pengcheng Laboratory
Shenzhen, China

Hanyue Zheng
Peking University
Beijing, China

Yao Xin
Guangzhou University
Guangzhou, China

Wenjun Li✉
Pengcheng Laboratory
Shenzhen, China

Tong Yang
Peking University
Beijing, China

Yangyang Wang
Tsinghua University
Beijing, China

Yang Xu
Fudan University
Shanghai, China

Weizhe Zhang
Harbin Institute of Technology
Shenzhen, China

Mingwei Xu
Tsinghua University
Beijing, China

Abstract

Sketch algorithms are crucial for identifying top-k items in large-scale data streams. Existing methods often compromise between performance and accuracy, unable to efficiently handle increasing data volumes with limited memory. We present Bubble Sketch, a compact algorithm that excels in both performance and accuracy. Bubble Sketch achieves this by (1) Recording only full keys of hot items, significantly reducing memory usage, and (2) Using *threshold relocation* to resolve conflicts, enhancing detection accuracy. Unlike traditional methods, Bubble Sketch eliminates the need for a Min-Heap, ensuring fast processing speeds. Experiments show Bubble Sketch outperforms the other seven algorithms compared, with the highest throughput and precision, and surpasses HeavyKeeper in accuracy by up to two orders of magnitude.

CCS Concepts

• **Information systems → Data stream mining**.

Keywords

Data stream; Approximate algorithm; Sketch; Top-k

ACM Reference Format:
Lu Cao, Qilong Shi, Yuxi Liu, Hanyue Zheng, Yao Xin, Wenjun Li✉, Tong Yang, Yangyang Wang, Yang Xu, Weizhe Zhang, and Mingwei Xu. 2024. Bubble Sketch: A High-performance and Memory-efficient Sketch for Finding Top-k Items in Data Streams. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM '24), October 21–25, 2024, Boise, ID, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3627673.3679882

∗Lu Cao conducted this work under the guidance of corresponding authors Wenjun Li (wenjunli@pku.org.cn) and Qilong Shi. This work was supported in part by the Major Key Project of Peng Cheng Lab (PCL2023A06), NSFC (62102203,62221003,62132004, 62372123), 173 Program (2021-JCJQ-JJ-0483). The source code is available on GitHub [1].

CIKM '24, October 21–25, 2024, Boise, ID, USA

ACM ISBN 979-8-4007-0436-9/24/10
https://doi.org/10.1145/3627673.3679882

1 Introduction

Detecting top-k frequent items in data streams is essential for applications like data mining [14, 16, 18, 36, 41, 42, 44, 45], traffic measurement [2, 6, 17, 32, 35, 46], and data security [3, 4, 8, 11, 12, 20]. For instance, data centers must detect significant data traffic items to balance load effectively, while social platforms detect frequent interactions to discern user connections [9, 19, 34, 38, 40].

In the big data era, the speed and volume of data streams make accurate item tracking challenging [22, 23]. Approximate solutions like sampling often lack accuracy. Advanced algorithms that operate on individual items need fast, consistent updates, avoiding slow DRAM in favor of limited-capacity SRAM [13, 39]. Sketch-based methods are popular for their efficiency in time and space with acceptable error margins [5, 7, 12, 24, 26, 31].

Top-k sketch algorithms are either *Min-Heap-based* or *Save-all-potentiality*. *Min-Heap-based* methods use a Min-Heap to track top-k items [29] by adding new items to the sketch and replacing the lowest frequency item in the heap if needed. This method is slow and memory-inefficient due to constant updates, as seen in HeavyKeeper [43], which needs $360 * k$ bits of memory. *Save-all-potentiality* methods store potential top-k items as <full key, frequency> pairs [25] in a single sketch, avoiding the Min-Heap but suffering from poor accuracy and high insertion overhead when memory is limited. Both methods struggle with performance and accuracy under tight memory constraints, leading to the question: ***Can we develop a compact sketch algorithm for top-k detection that excels in both speed and accuracy?***

We introduce **Bubble Sketch** (BS), which improves memory efficiency with a unique bucket structure and *threshold relocation* using *bubble sorting*. Key features of Bubble Sketch include:

(1) **Efficient Bucket Layout:** Buckets store "hot" entries as *<full key, frequency>* pairs and "cold" entries as smaller *<fingerprint, frequency>* pairs, saving memory.
(2) **Real-Time Bubble Sorting:** Entries are sorted in real-time, keeping hot items in the hot entries and exchanging keys and fingerprints as needed for efficient top-k queries.
(3) **Threshold Relocation for Accuracy:** Potential top-k item collisions are resolved by placing one in an alternate bucket, improving accuracy.

Implemented in C++ and tested on various datasets, Bubble Sketch shows superior performance, achieving the highest throughput and accuracy among current algorithms. It outperforms HeavyKeeper by two orders of magnitude in accuracy and is 1.5 times faster and more accurate than Waving Sketch.

2 Background

2.1 Problem Statement

Data Stream Model: As shown in Figure 1, a data stream $\mathcal{P} = \{p_1, p_2, \ldots, p_N\}$ contains N items and n distinct items. Items in data $\mathcal{P}$ can be categorized into n non-overlapping distinct items: $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$. The number of items in e_i is called the frequency of e_i (abbreviated as $e_i.f$ or f_i), and so we have $\sum_{i=1}^{n} f_i = N$. A distinct item also has a unique ID to identify itself, represented as $e_i.id$ (for example, we often use the 5-tuple headers to identify a TCP distinct packet in a network stream).

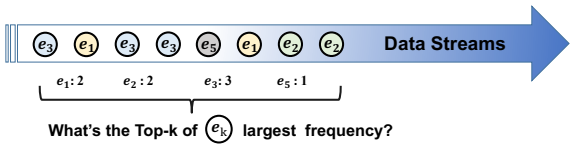


Figure 1: Finding Top-k Items.

Top-k Items Detection: Top-k items detection outputs the ID and frequency of the most frequent k items in a data stream, denoted as $(e_1, f_1), (e_2, f_2), \ldots, (e_k, f_k)$.

2.2 Related Work

Sketches are widely used for finding top-k items in data streams and are generally divided into Min-Heap-based and Save-all-potentiality-based solutions. Min-Heap-based methods use a Min-Heap to store top-k items. **CM Sketch + Min-Heap**[10] hashes items to multiple counters and updates the Min-Heap using the minimum counter value, which can misidentify cold items as hot and slows processing due to frequent updates. **HeavyKeeper** (HK) [43] and **Cuckoo Counter** (CC)[33] also suffer from throughput issues and high memory usage due to the Min-Heap.

Save-all-potentiality methods store all potential top-k items within the data structure. **Space-Saving** (SS)[30] replaces the smallest existing item with new items, but its accuracy is limited by not considering data stream characteristics. **Unbiased Space-Saving** (USS)[37] improves frequency estimation for multi-node scenarios but is less effective for single nodes. **Waving Sketch** [21] requires traversing all entries to extract top-k items and faces memory issues by storing complete IDs in each cell. **Double-Anonymous Sketch** (DAS) [47] aims for fairness but suffers from low throughput due to its two-layer structure.

2.3 Why Yet Another One?

Min-Heap-based algorithms, like Cuckoo Counter, achieve high precision but suffer from throughput issues. Save-all-potentiality algorithms, like Waving Sketch, lead to poor precision and high memory usage in tight memory scenarios. DAS improves precision but has low throughput due to its two-layer structure. To address these limitations, we propose Bubble Sketch , a single-layer algorithm that adapts to data skewness, reducing memory usage and enhancing precision without needing a Min-Heap. This design significantly boosts throughput, making BS approximately 8 times faster than DAS and the fastest among current top-k algorithms.

3 Bubble Sketch Framework

3.1 Data Structure

The Bubble Sketch data structure, illustrated in Figure 2, comprises two arrays, A_1 and A_2, designed to handle potential conflicts effectively. Each array has w buckets, and each bucket contains B entries of varying sizes, labeled {*entry_1*, *entry_2*, …, *entry_B*}. Entries are the basic units, and buckets serve as access units. The bit width of entries increases progressively, forming a ladder-like distribution. The largest entry, *entry_B*, stores hot items (top-k items) with an *ID* and a large *counter* for the *<full key, frequency>* pair. The smaller entries, designated for secondary-hot or cold items, contain a *fingerprint* and a smaller *counter* for the *<fingerprint, frequency>* pair. While the width of *ID* and *fingerprint* remains constant across entries, the counter width varies. This design leverages the skewness of data streams, optimizing storage by accommodating as many conflicting items as possible. The *real-time bubble sorting* technique dynamically relocates streams within the same bucket, ensuring hot streams occupy larger entries and cold streams smaller ones. Each bucket acts as a hash unit, requiring only O(1) access time and enabling linear traversal within the bucket. This design ensures high throughput during updates. Buckets are aligned with the machine word length (e.g., 128 bits) to maximize entry manipulation per memory access.

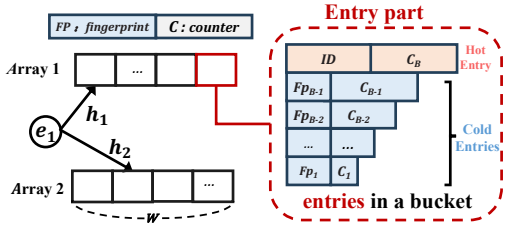


Figure 2: The data structure of Bubble Sketch.

3.2 Algorithm and Operations

For conciseness, we use $A_i[h][r]$ to represent the entry r of bucket h in array i. Initially, all entries are initialized to zero. Below we introduce the implementation process of the two main operations of our algorithm in turn: insertion and query.

3.2.1 *Insert.*

Each time an incoming item e is inserted, we first compute two hash values using formulas: $h_1 = hash_1(e)$ and $h_2 = hash_2(e)$, to identify two candidate buckets in two arrays, $A_1[h_1]$ and $A_2[h_2]$. The entries in these buckets are then traversed to find whether

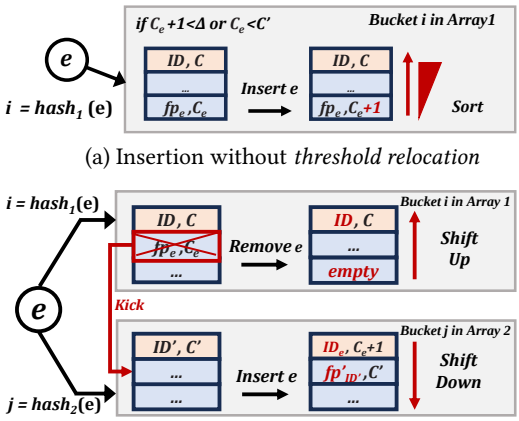


Figure 3: Insert matching items.

e already exists. If e is found in the bucket $A_1[h_1]$ as shown in Figure 3(a), its corresponding entry's counter C_e is incremented by 1. The entries within $A_1[h_1]$ are then sorted according to their counter values (frequencies), to make sure hot items are always at the top. When items switch between hot and cold entries, their IDs and fingerprints would be exchanged. If, after sorting, the updated entry becomes the second-largest entry and the value of $C_e + 1$ remains below the threshold Δ, the insertion process is complete.

If the updated frequency $C_e + 1$ exceeds the threshold Δ, it indicates that there may be two hot items conflicting within the bucket. Furthermore, if the frequency also surpasses the counter value C' of the topmost item in the alternative bucket $A_2[h_2]$, we initiate the relocation mechanism, as shown in Figure 3(b). The fingerprint of e is converted to ID and stored together with its frequency in the hot entry of $A_2[h_2]$. Additionally, the original items in $A_2[h_2]$ undergo a downward shift, where $A_2[h_2][B].ID$ is updated to $A_2[h_2][B-1].fp'_{ID'}$, and the item in $entry\_1$ is discarded.

If e is a new item (does not exist in the candidate bucket), it would be inserted into the first empty entry in either $A_1[h_1]$ or $A_2[h_2]$, and the associated counter is set to 1. In the event that both candidate buckets are full, we choose either $A_1[h_1][1]$ or $A_2[h_2][1]$ at random and decrement its counter by 1. Alternative replacement strategies, such as "Exponential-weakening decay" [43], "Probability replacement" [28, 37] may also be employed at this stage. If the frequency decays to 0, we insert e into the corresponding entry. Otherwise, e would be discarded.

The "*threshold relocation*" mechanism relies heavily on the value of Δ. To determine Δ, we use a global counter to dynamically monitor the maximum frequency, $f_{\max}$. According to the Zipf distribution, where α represents skewness and the frequency of the i-th largest item is $(1/i)^\alpha$ times that of the largest item, we set $\Delta = f_{\max} \times \left(\frac{1}{k}\right)^\alpha$ to identify the top-k items. Typically, $\alpha = 1$ is used. Higher α values decrease Δ, causing more frequent evictions.

3.2.2 Top-k Query.

For top-k queries, we only need to focus exclusively on the items contained in the hot entry ($entry\_B$). This is accomplished by looping through all hot entries in the array, subsequently sorting them, and then extracting the top-k items along with their frequency fields for reporting.

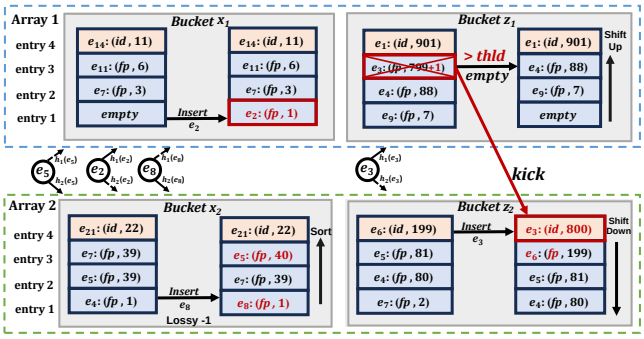


Figure 4: An running example of Bubble Sketch.

3.3 A Running Example

Figure 4 illustrates a running example of BS for insertion. The parameters are configured as follows: We assume $B = 4$ and $threshold = 799$. In this example, $entry_4$ in each bucket is the hot entry, and only the buckets involved in each array are displayed.

Case 1, to insert e_5: The two candidate buckets after hashing are x_1 and x_2. A search in these buckets reveals that e_5 already exists in entry $x_2[2]$. Consequently, $x_2[2].cnt$ is incremented by 1, and the entry is updated to $e_5 : (fp, 40)$. The whole bucket is then sorted in order. As a result, e_5 is shifted up to $x_2[3]$, while e_7 is shifted down to $x_2[2]$.

Case 2, to insert e_2: The item e_2 is not found in the corresponding buckets x_1 and x_2, but an empty entry exists at $x_1[1]$. Therefore, $e_2 : (fp, 1)$ is inserted into this entry.

Case 3, to insert e_8: The fingerprints in all entries do not match e_8. The smallest entry, $x_2[1]$, is selected (randomly). The counter is decayed to 0, and e_8 replaces e_4.

Case 4, to insert e_3: Searching through the two buckets, e_3 is found in entry $z_1[3]$. The count of e_3 is incremented by 1, updating the entry to $e_3 : (fp, 800)$. As e_3's frequency exceeds the threshold and surpasses the frequency in the top entry ($z_2[4]$) in the alternate bucket, $e_6 : (ID, 199)$ in entry $z_2[4]$ is shifted down to accommodate e_3. Item $e_3 : (ID, 800)$ is then placed into entry $A_2[z_2][4]$. Concurrently, item $e_6 : (ID, 199)$ is shifted to $entry_3$, becoming $e_6 : (fp, 199)$, with its ID replaced by a *fingerprint*, and all other existing entries are shifted down by one position. After $e_3 : (ID, 800)$ is relocated to $z_2[4]$ and the original entry $z_1[3]$ is cleared, the empty entry appears in $z_1[1]$ since the bucket needs to be sorted in ascending order.

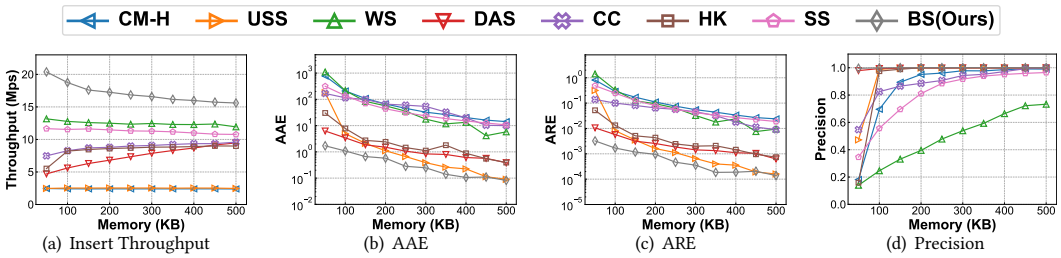
4 Performance Evaluation

4.1 Experiment Setup

The machine is Ubuntu 20.04 with an Intel i7-9700 CPU @ 3.0GHz and 16GB of DRAM. We use CAIDA Datasets obtained from the Equinix-Chicago data, as documented in [15], and align with the datasets in [42]. We consider the following metrics:

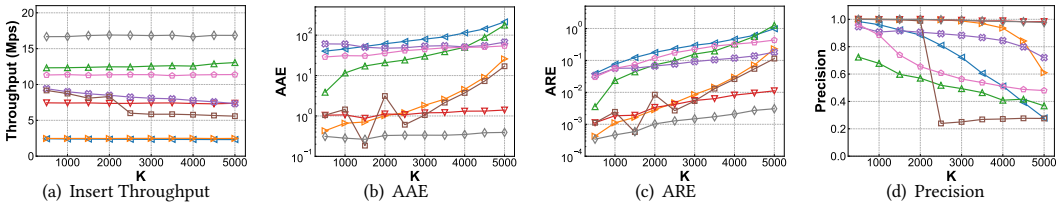
Throughput: It is calculated as N/T, where N is the total number of items processed, and T is the time taken to run the algorithm.

AAE: AAE is defined as $\frac{1}{|\Psi|}\sum_{(e_i \in \Psi)}\left|f_i - \tilde{f_i}\right|$. Here, f_i is the actual frequency of an item e_i, $\tilde{f_i}$ represents its estimated frequency, and Ψ is the estimated set of top-k items.



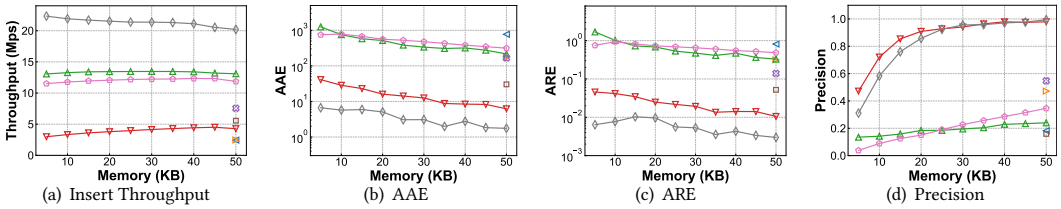
(a) Insert Throughput (b) AAE (c) ARE (d) Precision

Figure 5: Top-k estimation - vs. Memory - CAIDA.



(a) Insert Throughput (b) AAE (c) ARE (d) Precision

Figure 6: Top-k estimation - vs. k - CAIDA.



(a) Insert Throughput (b) AAE (c) ARE (d) Precision

Figure 7: Top-k estimation - vs. Tight memory.

ARE: ARE is defined as $\frac{1}{|\Psi|}\sum_{(e_i\in\Psi)}\left|f_i-\tilde{f}_i\right|/f_i$.

Precision: Precision (top-k) is defined as the proportion of accurately identified top-k items out of the total k items reported.

4.1.1 Experimental Configuration.

The Bubble Sketch (BS) algorithm is compared with CM sketch + Heap (CM-H) [10], Unbiased Space-Saving (USS) [37], Cuckoo Counter (CC) [33], Waving Sketch (WS) [21], HeavyKeeper (HK) [43], Lossy Counting (LC) [27], Double-Anonymous Sketch (DAS) [47], and Space-Saving (SS) [30] using their open-source codes.

4.2 Experiment Results

4.2.1 Experiments on Throughput. With Memory size ranges from 50KB to 500KB. Figures 5(a) show BS consistently surpasses other algorithms in throughput, even with small memory sizes like 50KB. BS achieves twice the throughput of WS and 2.5 times that of HK. With memory fixed at 250KB and k varying from 500 to 5000, Figures 6(a) show BS maintaining a distinct throughput advantage. BS's design avoids the Min-Heap structure, requiring fewer operations per insertion, achieving 2.3 times the throughput of CC.

4.2.2 Experiments on AAE. Figures 5(b) compare the AAE of different algorithms. BS maintains stable and low AAE values, almost an order of magnitude lower than HK and significantly better than USS within the memory range of 50KB to 500KB. Figures 6(b) show BS's AAE is superior across all k, maintaining low AAE even with increasing k. In contrast, HK and USS's performance deteriorates.

4.2.3 Experiments on ARE. Figure 5(c) shows that BS has lower error rates than CM-H in low-memory conditions. BS's stability is due to its full storage of item IDs, reducing error rates. Figures 6(c) show BS's ARE is significantly lower than DAS's and better than other sketches, except for HK at k of 1500.

4.2.4 Experiments on Precision. BS consistently achieves the highest precision across all memory scenarios. Figure 5(d) shows BS delivering 100% precision with small memory usage, outperforming other algorithms significantly. Figures 6(d) show BS achieving 100% precision as k increases from 500 to 5000, outperforming other algorithms whose precision declines with increasing k.

4.2.5 Experiments on Tight Memory. Experiments on the CAIDA dataset span memory ranges from 5KB to 50KB with k fixed at 1000. Figure 7 shows only BS, DAS, WS, and SS perform well in this range. BS achieves the highest throughput and lowest AAE and ARE in tight-memory scenarios, outperforming DAS, WS, and SS. Figure 7(d) shows BS's precision is lower than DAS below 20KB but converges to 1 within the 20KB to 50KB range.

5 Conclusion

Finding top-k hot items in data streams plays an important role in real-world applications. To comprehensively improve existing algorithms in terms of speed and accuracy, we propose a high-performance and memory-efficient algorithm called Bubble Sketch. It maintains the ordering and replacement of items within a bucket to achieve the effect of isolating hot and cold items. To verify its effectiveness, we deployed our sketch and 7 other sketches on the software platform and conducted the top-k items detection experiment. Extensive results show that our proposed Bubble Sketch has excellent accuracy/precision and the highest throughput.

References

[1] Our GitHub. https://github.com/wenjunpaper/BubbleSketch.

[2] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. 2021. SALSA: Self-Adjusting Lean Streaming Analytics. In *IEEE ICDE*.

[3] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. 2016. Heavy hitters in streams and sliding windows. In *IEEE INFOCOM*.

[4] Ran Ben Basat, Gil Einziger, Roy Friedman, Marcelo C Luizelli, and Erez Waisbard. 2017. Constant time updates in hierarchical heavy hitters. In *ACM SIGCOMM*.

[5] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2002. Finding frequent items in data streams. In *ICALP*.

[6] Aiyou Chen, Yu Jin, Jin Cao, and Li Erran Li. 2010. Tracking long duration flows in network traffic. In *IEEE INFOCOM*.

[7] Min Chen, Shigang Chen, and Zhiping Cai. 2016. Counter tree: A scalable counter architecture for per-flow traffic measurement. *IEEE/ACM Transactions on Networking* 25, 2 (2016), 1249–1262.

[8] Graham Cormode. 2011. Sketch techniques for approximate query processing. *Foundations and Trends in Databases. NOW publishers* (2011), 15.

[9] Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava. 2003. Finding hierarchical heavy hitters in data streams. In *ACM VLDB*.

[10] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[11] Garofalakis M Cormode G. 2005. Sketching streams through the net: Distributed approximate query tracking. In *ACM VLDB*.

[12] Haipeng Dai, Meng Li, Alex X Liu, Jiaqi Zheng, and Guihai Chen. 2019. Finding persistent items in distributed datasets. *IEEE/ACM Transactions on Networking* 28, 1 (2019), 1–14.

[13] Haipeng Dai, Yuankun Zhong, Alex X Liu, Wei Wang, and Meng Li. 2016. Noisy bloom filters for multi-set membership testing. In *ACM SIGMETRICS*.

[14] Zhuochen Fan, Ruixin Wang, Yalun Cai, Ruwen Zhang, Tong Yang, Yuhan Wu, Bin Cui, and Steve Uhlig. 2023. OneSketch: A Generic and Accurate Sketch for Data Streams. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2023), 12887–12901.

[15] Cooperative Association for Internet Data Analysis. 2016. *The CAIDA Traces*. http://www.caida.org/data/overview/

[16] Qun Huang, Xin Jin, Patrick PC Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. 2017. Sketchvisor: Robust network measurement for software packet processing. In *ACM SIGCOMM*.

[17] Qun Huang and Patrick PC Lee. 2014. LD-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams. In *IEEE INFOCOM*.

[18] Qun Huang, Patrick PC Lee, and Yungang Bao. 2018. Sketchlearn: Relieving user burdens in approximate measurement with automated statistical inference. In *ACM SIGCOMM*.

[19] Anukool Lakhina, Mark Crovella, and Christiphe Diot. 2004. Characterization of network-wide anomalies in traffic flows. In *ACM SIGCOMM*.

[20] Haoyu Li, Qizhi Chen, Yixin Zhang, Tong Yang, and Bin Cui. 2022. Stingy sketch: a sketch framework for accurate and fast frequency estimation. In *ACM VLDB*.

[21] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. 2020. Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams. In *ACM SIGKDD*.

[22] Meng Li, Deyi Chen, Haipeng Dai, Rongbiao Xie, Siqiang Luo, Rong Gu, Tong Yang, and Guihai ChenFellow. 2023. Seesaw Counting Filter: A Dynamic Filtering Framework for Vulnerable Negative Keys. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2023), 12987–13001.

[23] Meng Li, Rongbiao Xie, Deyi Chen, Haipeng Dai, Rong Gu, He Huang, Wanchun Dou, and Guihai Chen. 2023. A Pareto optimal Bloom filter family with hash adaptivity. *The VLDB Journal* 32, 3 (2023), 525–548.

[24] Tao Li, Shigang Chen, and Yibei Ling. 2012. Per-flow traffic measurement through randomized counter sharing. *IEEE/ACM Transactions on Networking* 20, 5 (2012), 1622–1634.

[25] Weihe Li and Paul Patras. 2023. Tight-Sketch: A High-Performance Sketch for Heavy Item-Oriented Data Stream Mining with Limited Memory Size. In *ACM CIKM*.

[26] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. 2016. FlowRadar: A Better NetFlow for Data Centers. In *USENIX NSDI*.

[27] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate frequency counts over data streams. In *ACM VLDB*.

[28] Motwani R. Manku G S. 2019. Randomized admission policy for efficient top-k, frequency, and volume estimation. *IEEE/ACM Transactions on Networking* 27, 4 (2019), 1432–1445.

[29] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2006. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems (TODS)* 31, 3 (2006), 1095–1133.

[30] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *ICDT*.

[31] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A Dinda, Ming-Yang Kao, and Gokhan Memik. 2007. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Transactions on Networking* 15, 5 (2007), 1059–1072.

[32] Qilong Shi, Chengjun Jia, Wenjun Li, Zaoxing Liu, Tong Yang, Jianan Ji, Gaogang Xie, Weizhe Zhang, and Minlan Yu. 2024. BitMatcher: Bit-level Counter Adjustment for Sketches. In *IEEE ICDE*.

[33] Qilong Shi, Yuchen Xu, Jiuhua Qi, Wenjun Li, Tong Yang, Yang Xu, and Yi Wang. 2023. Cuckoo Counter: Adaptive Structure of Counters for Accurate Frequency and Top-k Estimation. *IEEE/ACM Transactions on Networking* 31, 4 (2023), 1854–1869.

[34] Anirudh Sivaraman, Suvinay Subramanian, Mohammad Alizadeh, Sharad Chole, Shang-Tse Chuang, Anurag Agrawal, Hari Balakrishnan, Tom Edsall, Sachin Katti, and Nick McKeown. 2016. Programmable packet scheduling at line rate. In *ACM SIGCOMM*.

[35] Lu Tang, Qun Huang, and Patrick PC Lee. 2020. A fast and compact invertible sketch for network-wide heavy flow detection. *IEEE/ACM Transactions on Networking* 28, 5 (2020), 2350–2363.

[36] Lu Tang, Qun Huang, and Patrick PC Lee. 2020. SpreadSketch: Toward invertible and network-wide detection of superspreaders. In *IEEE INFOCOM*.

[37] Daniel Ting. 2018. Data sketches for disaggregated subset sum and frequent item estimation. In *ACM SIGMOD*.

[38] Yuhan Wu, Zhuochen Fan, Qilong Shi, Yixin Zhang, Tong Yang, Cheng Chen, Zheng Zhong, Junnan Li, Ariel Shtul, and Yaofeng Tu. 2022. She: A generic framework for data stream mining over sliding windows. In *ICPP*.

[39] Rongbiao Xie, Meng Li, Zheyu Miao, Rong Gu, He Huang, Haipeng Dai, and Guihai Chen. 2021. Hash adaptive bloom filter. In *IEEE ICDE*.

[40] Kaicheng Yang, Sheng Long, Qilong Shi, Yuanpeng Li, Zirui Liu, Yuhan Wu, Tong Yang, and Zhengyi Jia. 2023. SketchINT: Empowering INT with TowerSketch for per-flow per-switch measurement. *IEEE Transactions on Parallel and Distributed Systems* (2023).

[41] Tong Yang, Siang Gao, Zhouyi Sun, Yufei Wang, Yulong Shen, and Xiaoming Li. 2019. Diamond sketch: Accurate per-flow measurement for big streaming data. *IEEE Transactions on Parallel and Distributed Systems* 30, 12, 2650–2662.

[42] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *ACM SIGCOMM*.

[43] Tong Yang, Haowei Zhang, Jinyang Li, Junzhi Gong, Steve Uhlig, Shigang Chen, and Xiaoming Li. 2019. HeavyKeeper: an accurate algorithm for finding Top-k elephant flows. *IEEE/ACM Transactions on Networking* 27, 5 (2019), 1845–1858.

[44] Tong Yang, Yang Zhou, Hao Jin, Shigang Chen, and Xiaoming Li. 2017. Pyramid sketch: A sketch framework for frequency estimation of data streams. In *ACM VLDB*.

[45] Minlan Yu, Lavanya Jose, and Rui Miao. 2013. Software Defined Traffic Measurement with OpenSketch. In *USENIX NSDI*.

[46] Bohan Zhao, Xiang Li, Boyu Tian, Zhiyu Mei, and Wenfei Wu. 2021. DHS: Adaptive memory layout organization of sketch slots for fast and accurate data stream processing. In *ACM SIGKDD*.

[47] Yikai Zhao, Wenchen Han, Zheng Zhong, Yinda Zhang, Tong Yang, and Bin Cui. 2023. Double-Anonymous Sketch: Achieving Top-K-fairness for Finding Global Top-K Frequent Items. In *ACM SIGMOD*.