2FA Sketch: Two-Factor Armor Sketch for Accurate and Efficient Heavy Hitter Detection in Data Streams

Xilai Liu[1,2], Xinyi Zhang[2,3], Bingqing Liu[4], Tao Li[5(✉)], Tong Yang[6], and Gaogang Xie[2,3(✉)]

[1] Institute of Computing Technologies, Chinese Academy of Sciences
[2] University of Chinese Academy of Sciences, China   `liuxilai20121013@gmail.com`
[3] CNIC, Chinese Academy of Sciences, China   `{xyzhang,xie}@cnic.cn`
[4] Beijing University of Posts and Telecommunications, China   `bingqingliu9925@163.com`
[5] National University of Defense Technology, China   `taoli_network@163.com`
[6] Peking University, China   `yangtongemail@gmail.com`

Abstract. Detecting heavy hitters, which are flows exceeding a specified threshold, is crucial for network measurement, but it faces challenges due to increasing throughput and memory constraints. Existing sketch-based solutions, particularly those using Comparative Counter Voting, have limitations in efficiently identifying heavy hitters. This paper introduces the Two-Factor Armor (2FA) Sketch, a novel data structure designed to enhance heavy hitter detection in data streams. 2FA Sketch implements dual-layer protection through an improved `Arbitration` strategy for in-bucket competition and a cross-bucket conflict `Avoidance` hashing scheme. By theoretically deriving an optimal λ parameter and redesigning $vote^+_{new}$ as a conflict indicator, it optimizes the Comparative Counter Voting strategy. Experimental results show that 2FA Sketch outperforms the standard Elastic Sketch, reducing error rates by 2.5 to 19.7 times and increasing processing speed by 1.03 times.

Keywords: Heavy Hitter · Sketch · Data Stream.

1 Introduction

1.1 Motivation

Network measurement is a critical aspect of networking, providing essential data for applications such as load balancing, quality of service (QoS) management, network caching, congestion control, anomaly detection, and intrusion prevention [4, 10, 14, 20, 24, 32, 35]. Within this domain, identifying heavy hitters — flows whose sizes exceed a specified threshold — is particularly crucial as these flows often have a disproportionate impact on network performance and security [10, 34]. These flows are typically identified by a combination of five tuples: source IP/port, destination IP/port, and protocol.

However, heavy hitter detection faces significant challenges in modern networks. Detection systems must track millions of flows in real-time, updating counters for each packet arrival. The exponential growth in network throughput has outpaced advancements in CPU processing capabilities, while the memory wall — the growing disparity between processor and memory speeds — further exacerbates these limitations [21]. These factors collectively render traditional measurement approaches increasingly inadequate for contemporary network.

In recent years, sketch-based solutions have gained significant attention for addressing resource constraints by offering approximate methods balancing accuracy and efficiency [7–9,

11–13,16,19,28,31]. These approaches leverage SRAM's capabilities and simple computational operations to meet latency requirements for real-time processing, utilizing CPU power. They divide memory into key-value pairs to record heavy hitter information. However, SRAM's limited size (generally a few megabytes) [15] constrains the flows simultaneously monitored, challenging comprehensive capture of all heavy hitters.

Various sketch techniques aim to approach this goal by prioritizing heavy hitters during the allocation of limited memory resources, exploiting frequency disparities between heavy hitters and small flows [11, 17, 25, 28, 29, 33]. These techniques primarily rely on arbitration strategies during flow competition, which can be categorized into three main approaches: Exponential Weighted Decaying, Inverse Frequency Replacement, and Comparative Counter Voting (elaborated in Section 2). While recent advancements have optimized the first two strategies for more accurate heavy hitter detection, Comparative Counter Voting, as implemented in Elastic Sketch [28], has received less attention. Unlike probabilistic approaches, this deterministic arbitration strategy offers significant potential for enhancing heavy hitter detection, particularly in scenarios requiring predictability and consistency.

Elastic Sketch employs Comparative Counter Voting primarily to bifurcate flows into two structures: a Light Part with an 8-bit counter CM Sketch [6] for small flows, and a Heavy Part for larger flows, facilitating heavy hitter detection. This bifurcation enhances generality, but allocating memory to small flow data in the Light Part potentially constrains key-value pair storage for heavy hitters in the Heavy Part, suggesting room for optimization. However, naively discarding the Light Part would forfeit filtering advantages conferred by this structural separation, resulting in diminished generality without necessarily improving heavy hitter detection efficacy.

1.2   Our Solution

We introduce the **Two-Factor Armor (2FA) Sketch**, a novel data structure providing dual-layer protection for enhanced heavy hitter detection, addressing key limitations in existing Comparative Counter Voting approaches.

Comparative Counter Voting, which augments positive votes for matching flow keys and negative votes for mismatches, faces two critical challenges in its current implementation: (1) The empirical values of key parameters (λ and $vote^+_{new}$) are inherently linked to the Light Part in Elastic Sketch, making its simple removal ineffective. (2) The design lacks a mechanism to resolve competition among heavy hitters within a bucket, occasionally resulting in the loss of some heavy hitter records.

Our proposed solution addresses these challenges by theoretically deriving an optimal λ that preserves the flow order relationship, and redesigning $vote^+_{new}$ to function as an indicator of intense heavy hitter competition within a bucket. Upon detecting such competition, a rehashing mechanism is triggered for conflict avoidance, thereby preventing the omission of heavy hitters.

The main contributions of this paper are as follows:

- An improved `Arbitration` strategy for in-bucket competition, serving as the first armor layer. This Arbitration strategy, an improved version of Comparative Counter Voting, is applied to flows after the Light Part has been discarded to allocate more space for heavy hitters.
- A cross-bucket conflict `Avoidance` hashing scheme, acting as the second armor layer. This method complements the Arbitration strategy by addressing recall rate degradation caused by excessive flow congestion within individual buckets.

– Our experimental results demonstrate that 2FA Sketch significantly outperforms the standard Elastic Sketch, reducing the error rate by 2.5–19.7 times and increasing processing speed by 1.03 times. All source codes are publicly available on GitHub [1] to facilitate reproducibility and further research.

2   Background and Related Work

In this section, we begin with essential definitions of heavy hitters. We then briefly introduce the common sketch data structure.Finally, we focus our discussion on the distinguishing flow competing strategies within these sketches.

2.1   Problem Definition

Data streams are defined as continuous sequences of packets, each represented by a tuple (f, v_f) and processed exactly once. The flow key f comprises any combination of the five-tuple elements: source IP, destination IP, source port, destination port, and protocol number. The value v_f represents either a unit count 1 or the packet size.

A flow f is classified as a heavy hitter if its accumulated frequency V_f over the data stream exceeds the threshold $\Theta = \theta \cdot N$, where N denotes the total number of packets in the data stream and θ a predefined fraction, e.g, 0.01.

2.2   Common Sketch Data Structure

The common data structure of sketches for heavy hitter detection comprises a matrix of size $w \times d$, where d denotes the number of buckets, each containing w cells (as shown in Figure 1)[7]. This configuration aligns with set-associative cache memory structures. When $w = 1$, the structure is a hash table [23] with elevated collision probability, while $d = 1$ corresponds to the SpaceSaving [22] algorithm, necessitating Stream Summary technique for expedited cell location. These parameters primarily influence processing speed.

Each cell typically stores two fields for a flow: the flow key and its corresponding value. Packet arrival triggers mapping to a hashed bucket. Insertion is straightforward when the bucket contains the flow key or has empty cells. Conflicts arise when the hashed bucket is full and does not contain the flow key. To address this, various methods identify the cell with the minimum value C within the bucket, minimizing conflict. The incoming flow key then undergoes arbitration with the existing flow to determine retention and resulting value. Exploiting the frequency disparity between heavy hitter and small flows, research has yielded three main arbitration strategies: Exponential Weighted Decaying [27, 29], Inverse Frequency Replacement [3, 11, 25, 33], and Comparative Counter Voting [28]. The configuration of these strategies predominantly affects processing accuracy.

---

[7] Here, structures containing key-value pairs are generally referred to as counter-based methods in previous research, while those containing only values are termed sketch-based methods. For simplicity, we refer to both as "sketches" in this paper. The latter type of methods is discussed in Section 2.6.
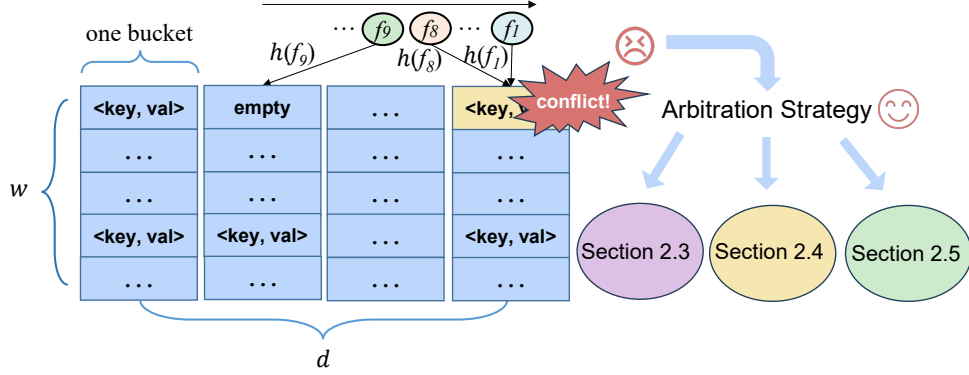
Fig. 1: Common Sketch Data Structure.

2.3 Exponential Weighted Decaying

This arbitration strategy was initially introduced in the HeavyGuardian paper [27]. For the cell with the minimum value C, the count field is decayed by 1 with probability $P = b^{-C}$, where b is a predefined constant (e.g., $b = 1.08$). Post-decay, if the count field reaches 0, the new flow f replaces the flow key field and resets the count to 1; otherwise, f is either discarded or inserted into an alternative structure. This approach amplifies the value differentials between conflicting flows by utilizing the value as an exponent. The HeavyKeeper [29] algorithm also incorporated this concept. Subsequently, ActiveKeeper [26] and HeavyTracker [30] have further refined the probability expression, building upon HeavyKeeper to enhance accuracy. This strategy is especially suitable to the top-k elephant flow detection task.

2.4 Inverse Frequent Replacement

This approach probabilizes replacement while maintaining deterministic counting, in contrast to the previous method that probabilized counting and determinized replacement. For the cell with the minimum value C, counting invariably occurs upon the arrival of a new flow, incrementing by 1. The key field has a $\frac{1}{C+1}$ probability of being replaced by the new flow and a $\frac{C}{C+1}$ probability of remaining unchanged. This method amplifies the tracking differences between flows by utilizing the value in the denominator as the replacement probability. Several algorithms, including USS, CocoSketch, RAP, and ChainSketch [3, 11, 25, 33], have implemented this design. These studies demonstrate that under this replacement probability, the expected count of a flow remains unbiased.

2.5 Comparative Counter Voting

In this strategy, each bucket comprises multiple key-value cells (with values serving as $vote^+$) and one final cell without a key, which records negative votes ($vote^-$), collectively forming the Heavy Part in Elastic Sketch. Upon conflict in the minimum-value cell, $vote^-$ increments while $vote^+$ remains static. Replacement occurs when $\frac{vote^-}{vote^+}$ exceeds a preset threshold λ (e.g., 4). During replacement in the Elastic Sketch, $vote^+_{new}$ resets to 1, $vote^-$ to 0, and the flow key updates to the new flow (f). The original flow's count is then evicted to the Light Part, a

CM Sketch. Thus, large flow counts combine the cell's value and potential CM Sketch counts. This approach, utilizing vote comparison between different count values, ensures deterministic replacement and counting mechanisms.

2.6   Other Sketches for Heavy Hitters

Various methods beyond key-value pair-based sketches are employed to identify heavy hitters. Algorithms that eschew key storage, such as CM [6], CU [10], Count [5], and CSM [18] sketches, often utilize an auxiliary min-heap for heavy hitter maintenance. Among these, the CU sketch achieves the highest accuracy, while the CSM sketch provides optimal processing speed.

3   The 2FA Sketch

In this section, we present the details of 2FA Sketch, including its data structure and operations, design innovations and examples.

3.1   Algorithm Overview

Data Structure (Fig. 2): The data structure of 2FA Sketch aligns with the structure outlined in Section 2.2, retaining only the Heavy Part of Elastic Sketch. It comprises a bucket array B of size d, determined by the available memory and required accuracy constraints. Each bucket contains w cells: one cell records the negative votes ($vote^-$) for the bucket, while the remaining cells store two fields per flow: the flow key and positive votes ($vote^+$, equivalent to the value field).
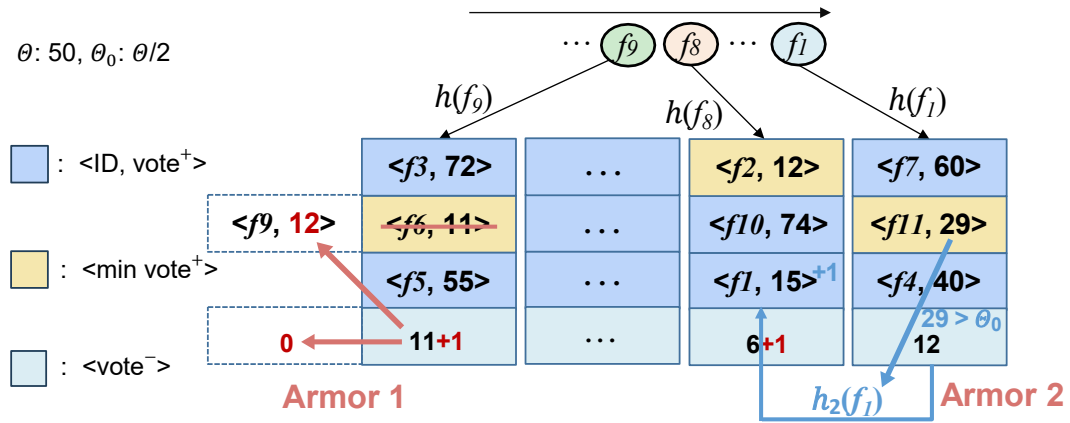

Fig. 2: An example for the 2FA Sketch.

Insertion (Fig. 2): When inserting an incoming packet with flow key f, we calculate the hash function, check the hashed bucket $B[h(f)]$ and do the corresponding operations. The pseudo-code without SIMD (Single Instruction Multiple Data) speedup is shown in algorithm 1. After locating the hashed bucket $B[h(f)]$:

1) If f matches a flow key within any cell of the bucket, only the corresponding positive votes ($vote^+$) are incremented.

2) If f is not in the bucket, there are the following two cases:

   Case 1: There are empty cells in the bucket. We just insert f with $vote^+$ of 1 into the first empty cell;

   Case 2: When no empty cell exists in the bucket, we first increment $B[h(f)].vote^-$ by 1 (hereafter, $B[h(f)]$ is omitted for brevity). We then locate the cell $[min_{index}]$ with the minimum positive votes in the bucket.

   1) If $[min_{index}].vote^+$ exceeds Θ_0 (a threshold related to Θ, e.g., $\Theta/2$), we rehash to an alternative backup bucket $B[h_2(f)]$ and restart the insertion process (only once).

   2) Otherwise, we compare $vote^-$ with $[min_{index}].vote^+$. If $\frac{vote^-}{[min_{index}].vote^+} > \lambda = 1$, $[min_{index}].key$ is replaced by f and $[min_{index}].vote^+$ (equivalent to $vote^+_{new}$) replaced by $vote^-$; otherwise, f is discarded.

Algorithm 1: Insertion Procedure

Input: an incoming flow f, threshold Θ

1 $FP = \text{hash}(f)$, $pos = FP \% d$, $min_{val} = \infty$, $min_{index} = 0$, $\Theta_0 = \Theta/2$, $\lambda = 1$

2 **for** $i \leftarrow 1$ **to** $w - 1$ **do**

3   **if** $FP == B[pos].key[i]$ **then**

4          $B[pos].vote[i]+ = 1$ // $vote[i]$ represents $vote^+$ in loop

5   return

6   **if** $B[pos].val[i] < min_{val}$ **then**

7          $min_{index} = i$

8          $min_{val} = B[pos].vote[i]$

9 **if** $min_{val} == 0$ **then**   // Case 1

10     $B[pos].key[min_{index}] = f$, $B[pos].vote[min_{index}] = 1$

11   return

12 **if** $min_{val} >= \Theta_0$ **then**   // Case 2.1, $vote^+_{new}$'s judgment

13     $hash = h_2$, goto line 1

14 $guard_{val} = B[pos].vote[0] + 1$ // $vote[0]$ represents $vote^-$

15 **if** $\frac{guard_{val}}{min_{val}} >= \lambda$ **then**   // Case 2.2

16     $B[pos].vote[0] = guard_{val}$

17 **else**

18     $B[pos].key[min_{index}] = f$, $B[pos].vote[0] = 0$

19     $B[pos].vote[min_{index}] = guard_{val}$ // $vote^+_{new}$'s update

---

Query: For a single flow f, querying whether it is a heavy hitter follows a process similar to insertion. The algorithm locates the buckets $B[h(f)]$ and $B[h_2(f)]$ using hash functions. If f matches a flow key within any cell of these buckets, we obtain the corresponding $vote^+$ value. The flow is reported as a heavy hitter if this value exceeds the threshold Θ. Notably, the information for f is stored in at most one of the buckets $B[h(f)]$ or $B[h_2(f)]$.

To identify all heavy hitters, the algorithm traverses all cells within each bucket, querying the size of each flow. Any flow with a size surpassing the predetermined threshold is classified and reported as a heavy hitter.

3.2   Design Innovations

Armor 1 (Fig. 2): Our first Armor layer is an improved `Arbitration` strategy for in-bucket competition, focusing on two key parameters. The assignment of $vote_{new}^+$ determines the initial frequency count for a new flow during replacement. We have $vote_{new}^+ \leq vote^-$, with equality possible only when two flows compete within a single cell. Therefore, we set $vote_{new}^+$ to $vote^-$ instead of 0 to ensure that any flow recorded in the bucket is not underestimated.

The parameter λ governs flow eviction criteria. To preserve flow order relationships, λ must not be less than the smallest bucket, ensuring only keys with higher frequency counts in the bucket are recorded (i.e., $\lambda \geq 1$). We set λ to the boundary value of 1 as it allows for quick incorporation of potential heavy hitters when $vote^-$ grows rapidly to $[min_{index}].vote^+$, indicating the presence of keys with large frequencies that don't match multiple cells. Elastic's more lenient judgment on this point is due to its ability to count potentially evicted large flows in its Light Part, making it less sensitive to λ variations at the cost of relatively larger errors for heavy hitters.

Armor 2 (Fig. 2): Our second Armor layer implements a cross-bucket conflict `Avoidance` hashing scheme. This approach is founded on two key observations:

1. When memory is constrained, hash imbalances and limited bucket numbers can lead to disproportionate flow mappings, resulting in some buckets exceeding their cell capacity while others remain underutilized. This issue cannot be resolved through the first Armor layer's arbitration strategy alone.
2. The $vote_{new}^+$ value from the arbitration strategy allows us to assess the degree of conflict within a bucket. Throughout the insertion process, $vote_{new}^+$ does not decrease in the replacements, causing the corresponding $[min_{index}].vote^+$ to consistently increase. This metric effectively gauges the intensity of multi-cell conflicts within a bucket.

We recommend the threshold Θ_0 between $0.4\Theta \sim 0.8\Theta$. When $[min_{index}].vote^+$ exceeds this threshold, we employ an alternative hash function to redirect to a backup bucket, thereby achieving conflict avoidance.

Examples (Fig. 2): We use three examples to show the differences between the standard Elastic and our 2FA Sketch in terms of the insertion operation. Given an incoming packet with flow key f_1, the hash function $h(f_1)$ is calculated and the hashed bucket is found. Since flow f_1 is not stored in the bucket and $[min_{index}].vote^+ > \Theta_0$ (Θ_0 sets to $\frac{\Theta}{2}$ here), we rehash to another bucket. At this time, it matches one cell luckily and the corresponding $vote^+$ increases by 1.

For another incoming packet with flow key f_8, we calculate the hash function and find the hashed bucket. Since f_8 is not stored in the bucket, $vote^-(= 6)$ is incremented by one. We then find the flow with the smallest size, f_1, and find that $vote^-$ is not larger than $vote^+(= 11)$ of f_1. Therefore, f_8 is discarded and nothing else is changed.

For another flow f_9, we calculate the hash function and find the hashed bucket. We find flow f_6, the flow with the smallest value in the bucket. Because $vote^-(= 12 = 11+1)$ is larger than $vote^+(= 11)$ of f_6, f_6 is replaced by f_9, and the size of f_9 is set to $vote^-$ (12, *i.e.*, $vote^+$ of f_6 plus 1). Flow f_6 is discarded and $vote^-$ is set to zero.

4   Experimental Results

We present experimental results comparing 2FA Sketch with state-of-the-art heavy hitter detection algorithms. Section 4.1 describes the experimental setup, followed by performance and parameter evaluations in Section 4.2 and Section 4.3.

4.1   Experimental Setup

Datasets: We have employed two datasets for our study.
CAIDA Dataset: We use the public traffic trace datasets collected in Equinix-Chicago monitor from CAIDA [2]. In our experiments, each packet is identified by its source IP address (4 bytes). There are about 25 millions packets in total.
Synthetic Dataset: We generate synthetic datasets with 30 millions packets each, following the **Zipf** distribution $p(x) = \frac{x^{-\alpha}}{\zeta(\alpha)}$, with varying α values ranging from 0.4 to 1.2 with a step of 0.2. Zipf's law, an empirical observation, describes the distribution in phenomena such as the frequency of web page visits. The different α values represent varying levels of skewness, allowing us to simulate data streams with diverse characteristics.
Implementation: We compare six heavy hitter detection approaches: SpaceSaving (SS) [22], Count/CM Sketch [5,6] with a min-heap (CountHeap/CMHeap), Elastic Sketch [28], Chain Sketch (Chain) [11] and 2FA Sketch. All algorithms are implemented in C++ with 100KB default memory size and 0.01% heavy hitter threshold in every experiment. For Elastic and Chain Sketch, we use the parameters in the open-sourced code. For both Elastic and 2FA Sketch, we store 7 flows and one $vote^-$ in each bucket of the Heavy Part. Elastic uses a 3:1 Heavy to Light Part ratio. The variant of 2FA Sketch employing solely Armor 1 is designated as 1FA. In the implementation of Armor 2, the ratio $\frac{\Theta_0}{\Theta}$ is configured to 0.5. For CountHeap / CMHeap, we use 3 hash functions for the sketch and set the heap capacity to 4096 nodes [28].
Computation Platform: we conducted all the experiments on a machine with one 4-core processor (4 threads, AMD EPYC 7662@2.0GHz) and 15.4 GB DRAM memory. To accelerate the processing speed, we use SIMD (Single Instruction Multiple Data) instructions for both Elastic and 2FA Sketch. With the AVX2 instruction set, we can find the minimum counter and its index among 8 counters in a single comparison instruction. Also, we can compare 8 32-bit integers with another set of 8 32-bit integers in a single instruction.
Metrics: We use the following seven metrics to evaluate the performance of compared algorithms.
1) AAE (Average Absolute Error): $\frac{1}{|\Phi|} \sum_{e_i \in \Phi} |f_i - \hat{f}_i|$, where Φ is the query set of heavy flows and f_i and $\hat{f}_i$ are the actual and estimated flow sizes of flow f_i, respectively.
2) ARE (Average Relative Error): $\frac{1}{|\Phi|} \sum_{e_i \in \Phi} \frac{|f_i - \hat{f}_i|}{f_i}$, where Φ, f_i, and $\hat{f}_i$ are defined above.
3) PR (Precision Rate): Ratio of the number of correctly reported flows to the number of reported flows.
4) RR (Recall Rate): Ratio of the number of correctly reported flows to the number of true flows.
5) F_1 score: $\frac{2 \times PR \times RR}{PR + RR}$, where PR and RR are defined above.
6) Rehashing Ratio: The proportion of non-matching packets that are rehashed to alternative buckets.
7) Throughput: million packets per second (Mpps). We use it to evaluate the processing speed of different sketches. The experiments on processing speed are repeated 100 times to minimize accidental errors.

4.2   Evaluation on Accuracy and Processing Speed

Accuracy evaluation on CAIDA (Figure 3a-3d): With memory sizes of 100KB-500KB, 2FA Sketch outperforms Chain and Elastic in accuracy. 2FA Sketch achieves $7.3 \sim 8.1$ times

lower AAE and $5.7 \sim 7.5$ times lower ARE than Elastic on CAIDA. 2FA Sketch, Elastic, and Chainsketch maintain nearly 100% PR and RR even at 100KB memory. The other methods, requiring additional memory for min-heap or stream summary structures, perform poorly here but are more suited for top-k elephant flow detection. Subsequently, we evaluate with further reduced memory for 2FA Sketch, Elastic and Chainsketch.
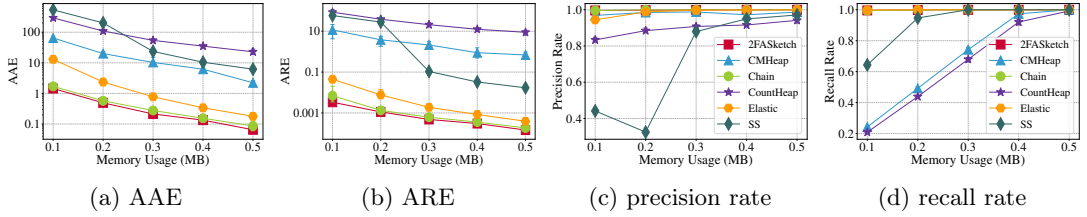

(a) AAE   (b) ARE   (c) precision rate   (d) recall rate

Fig. 3: Heavy Hitter Detection Accuracy Comparison: CAIDA Dataset (3a and 3b share the same legend as the others)

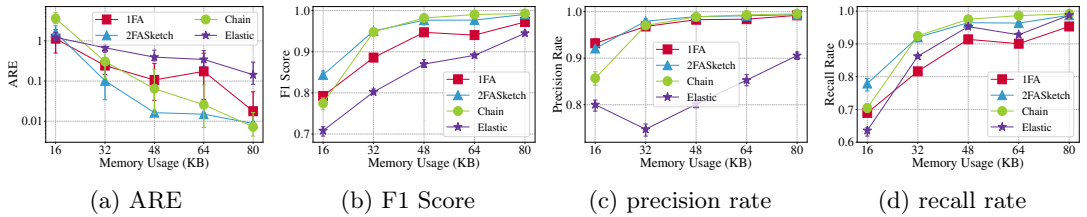(a) ARE   (b) F1 Score   (c) precision rate   (d) recall rate

Fig. 4: Heavy Hitter Detection Accuracy Comparison: CAIDA Dataset Under Memory Constraints

Accuracy evaluation on CAIDA under memory constraints (Figure 4a-4d): The 2FA Sketch outperforms other methods across all metrics, even with limited memory sizes ranging from 16KB to 80KB. Notably, it achieves an F1 score of approximately 85% with only 16KB of memory. This performance can be attributed to the significant improvement (over 10%) of Armor 2 over Armor 1, which was achieved without compromising the ARE and PR for detected flows.
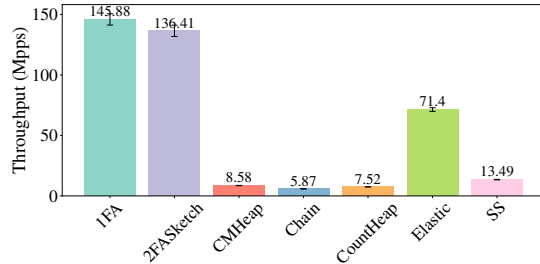

Fig. 5: Throughput Comparison on CAIDA Dataset, including 1FA

Throughput (Figure 5): Our experiments show that 2FA Sketch significantly outperforms five other algorithms on our CPU platform. It achieves nearly 140 Mpps with SIMD optimization, about 1.03 times faster than Elastic and over 9 times faster than conventional algorithms limited to 15 Mpps. Notably, Armor 2's rehashing, triggered only during intense conflicts, reduces throughput by just 6.5% compared to 1FA.

4.3   Evaluation on Parameter Settings

Accuracy evaluation on 2FA Sketch with different skewness α (Figure 6a-6d): Our results demonstrate that 2FA Sketch maintains an F1 Score above 85% with only 8KB of memory, but exhibits reduced performance at 4KB. Data streams with moderate skewness prove more challenging, characterized by a high number of flows and intense competition between large and small flows.

AAE and ARE comparison of 2FA Sketch with different λ (Figure 7a-7b): Our experimental results show that 2FA Sketch achieves highest accuracy at $\lambda = 1$, with performance improving as λ decreases. While Elastic outperforms 2FA Sketch at $\lambda = 8$, it fails to improve at $\lambda = 1$, where 2FA Sketch excels. At $\lambda = 1$, 2FA Sketch achieves $4.4 \sim 8.2$ times smaller AAE and $4.4 \sim 6.9$ times smaller ARE compared to Elastic. These experiments were conducted with memory sizes ranging from 100KB to 500KB.
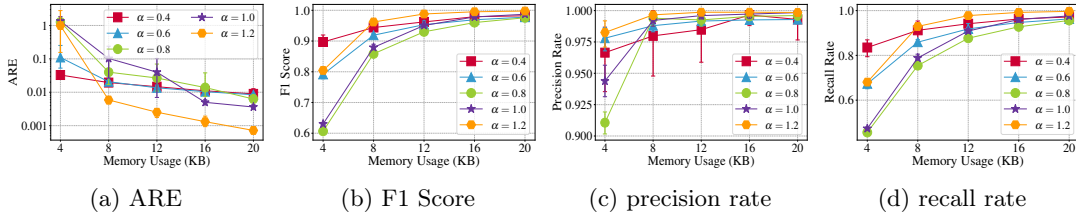

(a) ARE   (b) F1 Score   (c) precision rate   (d) recall rate

Fig. 6: Performance Analysis of 2FA Sketch with Varying Skewness α on Synthetic Dataset

(a) AAE   (b) ARE   (c) Threshold Ratio vs.   (d) Threshold Ratio vs.
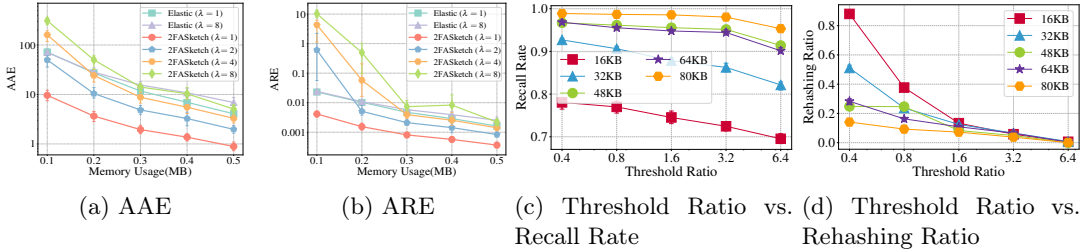   Recall Rate   Rehashing Ratio

Fig. 7: Performance Analysis of 2FA Sketch with Varying λ and $\frac{\Theta_0}{\Theta}$ Ratios on CAIDA Dataset

RR evaluation with different threshold ratios(Figure 7c-7d): Given Armor 2's significant improvement in RR shown in Figure 4d, we assessed the impact of varying $\frac{\Theta_0}{\Theta}$ on RR. Optimal RR is achieved with threshold ratios between 0.4 and 0.8, with a more pronounced effect under memory constraints. Considering the threshold ratio's influence on throughput, we also evaluated the rehashing ratio, observing a substantial decrease within the 0.4-0.8 range.

5   Conclusion

This paper introduced the Two-Factor Armor (2FA) Sketch, a novel data structure designed to enhance heavy hitter detection in high-throughput network environments. By addressing key limitations in existing Comparative Counter Voting approaches, 2FA Sketch provides dual-layer protection for improved accuracy and efficiency. The first layer, an improved Arbitration strategy, optimizes in-bucket competition through a theoretically derived optimal λ parameter and a redesigned $vote_{new}^+$ as a conflict indicator. The second layer, a cross-bucket conflict Avoidance hashing scheme, further enhances heavy hitter detection by mitigating recall rate degradation caused by flow congestion. Experimental results demonstrate the significant performance improvements of 2FA Sketch over the standard Elastic Sketch, with error rates reduced by 2.5 to 19.7 times and processing speed increased by 1.03 times.

References

1. The source codes of our and other related algorithms. `https://github.com/2FA-Sketch/2FA-Sketch`
2. The CAIDA Anonymized Internet Traces. `http://www.caida.org/data/overview/`
3. Basat, R.B., Einziger, G., Friedman, R., Kassner, Y.: Randomized admission policy for efficient top-k and frequency estimation. In: Proc. IEEE INFOCOM. pp. 1–9. IEEE (2017)
4. CAI, W., YANG, S., SUN, G., ZHANG, Q., YU, H.: Adaptive load balancing for parameter servers in distributed machine learning over heterogeneous networks. ZTE Communications **21**(1), 72 (2023)
5. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. Automata, languages and programming (2002)
6. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. Journal of Algorithms **55**(1) (2005)
7. Dai, H., Meng, L., Liu, A.X.: Finding persistent items in distributed, datasets. In: Proc. IEEE INFOCOM (2018)
8. Dai, H., Shahzad, M., Liu, A.X., Zhong, Y.: Finding persistent items in data streams. Proceedings of the VLDB Endowment **10**(4), 289–300 (2016)
9. Dai, H., Zhong, Y., Liu, A.X., Wang, W., Li, M.: Noisy bloom filters for multi-set membership testing. In: Proc. ACM SIGMETRICS. pp. 139–151 (2016)
10. Estan, C., Varghese, G.: New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. ACM Transactions on Computer Systems (TOCS) **21**(3) (2003)
11. Huang, J., Zhang, W., Li, Y., Li, L., Li, Z., Ye, J., Wang, J.: Chainsketch: An efficient and accurate sketch for heavy flow detection. IEEE/ACM Transactions on Networking **31**(2), 738–753 (2022)
12. Huang, Q., Jin, X., Lee, P.P., Li, R., Tang, L., Chen, Y.C., Zhang, G.: Sketchvisor: Robust network measurement for software packet processing. In: Proceedings of the 2017 ACM SIGCOMM Conference. ACM (2017)
13. Huang, Q., Lee, P.P.: Ld-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams. In: Proc. IEEE INFOCOM. pp. 1420–1428. IEEE (2014)
14. JI, Y., HAN, J., ZHAO, Y., ZHANG, S., GONG, Z.: Log anomaly detection through gpt-2 for large scale systems. ZTE Communications **21**(3), 70 (2023)

15. Kim, D., Zhu, Y., Kim, C., Lee, J., Seshan, S.: Generic external memory for switch data planes. In: Proceedings of the 17th ACM Workshop on Hot Topics in Networks. pp. 1–7 (2018)
16. Li, D., Cui, H., Hu, Y., Xia, Y., Wang, X.: Scalable data center multicast using multi-class bloom filter. In: 2011 19th IEEE International Conference on Network Protocols. pp. 266–275. IEEE (2011)
17. Li, S., Huang, J., Zhang, W., Shao, J.: PA-Sketch: A fast and accurate sketch for differentiated flow estimation. In: 2023 IEEE 31st International Conference on Network Protocols. pp. 1–11. IEEE (2023)
18. Li, T., Chen, S., Ling, Y.: Per-flow traffic measurement through randomized counter sharing. IEEE/ACM Transactions on Networking **20**(5) (2012)
19. Li, Y., Miao, R., Kim, C., Yu, M.: Flowradar: A better netflow for data centers. In: Proc. NSDI (2016)
20. Lin, W., Li, C., Xu, L., Xie, K.: Enhanced network traffic anomaly detection: Integration of tensor eigenvector centrality with low-rank recovery models. IEEE Transactions on Services Computing (2024)
21. McKee, S.A.: Reflections on the memory wall. In: Proceedings of the 1st conference on Computing frontiers. p. 162 (2004)
22. Metwally, A., Agrawal, D., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: Proc. Springer ICDT (2005)
23. Scouarnec, N.L.: Cuckoo++ hash tables: High-performance hash tables for networking applications. In: Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems. pp. 41–54 (2018)
24. Shroff, N., et al.: A measurement-analytic approach for qos estimation in a network based on the dominant time scale. IEEE/ACM Transactions on Networking **11**(2), 222–235 (2003)
25. Ting, D.: Data sketches for disaggregated subset sum and frequent item estimation. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1129–1140 (2018)
26. Wu, M., Huang, H., Sun, Y.E., Du, Y., Chen, S., Gao, G.: Activekeeper: an accurate and efficient algorithm for finding top-k elephant flows. IEEE Communications Letters **25**(8), 2545–2549 (2021)
27. Yang, T., Gong, J., Zhang, H., Zou, L., Shi, L., Li, X.: Heavyguardian: Separate and guard hot items in data streams. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. pp. 2584–2593 (2018)
28. Yang, T., Jiang, J., Liu, P., Huang, Q., Gong, J., Zhou, Y., Miao, R., Li, X., Uhlig, S.: Elastic sketch: Adaptive and fast network-wide measurements. In: Proceedings of the 2018 ACM SIGCOMM Conference. ACM (2018)
29. Yang, T., Zhang, H., Li, J., Gong, J., Uhlig, S., Chen, S., Li, X.: Heavykeeper: an accurate algorithm for finding top-k elephant flows. IEEE/ACM Transactions on Networking **27**(5), 1845–1858 (2019)
30. Yu, J., Sun, Y.E., Huang, H., Du, Y., Gao, G., Xu, H., Chen, S.: Heavytracker: An efficient algorithm for heavy-hitter detection in high-speed networks. In: 2022 IEEE 28th International Conference on Parallel and Distributed Systems (ICPADS). pp. 362–370. IEEE (2023)
31. Yu, M., Jose, L., Miao, R.: Software defined traffic measurement with opensketch. In: Proc. NSDI. vol. 13 (2013)
32. ZHANG, Q., HAN, J., Cheng, L., ZHANG, B., GONG, Z.: Approach to anomaly detection in microservice system with multi-source data streams. ZTE Communications **20**(3), 85–92 (2022)
33. Zhang, Y., Liu, Z., Wang, R., Yang, T., Li, J., Miao, R., Liu, P., Zhang, R., Jiang, J.: Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. In: Proceedings of the 2021 ACM SIGCOMM Conference. pp. 207–222 (2021)
34. Zhang, Y., Fang, B., Zhang, Y.: Identifying heavy hitters in high-speed network monitoring. Science China Information Sciences **53**, 659–676 (2010)
35. Zheng, H., Huang, C., Han, X., Zheng, J., Wang, X., Tian, C., Dou, W., Chen, G.: μmon: Empowering microsecond-level network monitoring with wavelets. In: Proceedings of the 2024 ACM SIGCOMM Conference. pp. 274–290 (2024)