RA-Sketch: A Unified Framework for Rapid and Accurate Sketch Configurations

Kejun Guo[†], Fuliang Li[†✉], Yuting Liu[†], Jiaxing Shen[‡], Xingwei Wang[†✉]
[†]Northeastern University, Shenyang 110819, China, [‡]Lingnan University, Hong Kong
Email: kejunguo@163.com, lifuliang@cse.neu.edu.cn, 2301921@stu.neu.edu.cn, jiaxingshen@LN.edu.hk, wangxw@mail.neu.edu.cn

***Abstract*—Network measurement sketches enable efficient traffic monitoring but require careful parameter configuration to balance accuracy and memory efficiency. We present *RA-Sketch*, a framework for generating memory-optimal sketch configurations that satisfy user-defined error constraints across diverse network measurement tasks. Unlike existing approaches that rely on computationally intensive experimental testing, RA-Sketch introduces: 1) *Poisson-distributed collision modeling* to construct error predictors for both frequency-independent tasks (membership query, heavy-hitter detection) and frequency-dependent tasks (frequency/cardinality estimation), eliminating the need for empirical validation; 2) A *hierarchical search strategy* combining power-of-two scaling and binary search, reducing iterations through optimized parameter initialization. RA-Sketch supports 10+ sketch architectures including Bloom Filter, Elastic Sketch, HeavyGuardian, HeavyKeeper, CM/CO Sketch, gSkt, rSkt1 and so on. Evaluations on real-world network traces demonstrate: 1) 6–7 orders of magnitude faster configuration than benchmark-based methods; 2) Prediction errors $\leq$10% for heavy-hitter detection, while prediction errors for membership query, and frequency/cardinality estimation are close to zero; 3) Memory utilization approaches theoretical minima. The framework's generality and efficiency enable real-time reconfiguration of sketches under dynamic network conditions.**

***Index Terms*—sketch, error estimation, network measurement.**

I. Introduction

A. Background and Motivation

Network measurement plays a critical role in network management and optimization [1]–[10]. Typically, network measurement system monitors application traffic to provide insights into tasks such as membership query, heavy-hitter detection, frequency/cardinality estimation and so on. Recent advances leverage sketches, approximate measurement algorithms, to achieve high accuracy and low resource overhead.

Most existing sketches are limited to approximate queries. However, in many scenarios, users require sketch configurations that meet user-defined error constraints while minimizing memory overhead. These error constraints vary depending on the specific task. For instance, in membership query, the error constraint may be defined as false positive rate (FPR) $\leq 1\%$, where FPR represents the proportion of flows falsely reported as present in the sketch. In heavy-hitter detection, the error constraint may be defined as recall rate (RR) $\geq 90\%$, where RR denotes the proportion of true heavy hitters reported by the sketch to all true heavy hitters. For frequency estimation and cardinality estimation, the error constraint may be defined as average absolute error (AAE) ≤ 10, where AAE is the mean of the absolute differences between the sketch query values and the true values across all flows.

979-8-3315-0376-5/25/$31.00 ©2025 IEEE

TABLE I: Comparison of existing solutions and ideal solution.

Solutions/Advantages	General	Rapid	Accurate
Theoretical	×	√	×
Simulation-based	×	√	√
Benchmark-based	√	×	√
RA-Sketch	√	√	√

Existing sketch configuration solutions can be classified into three categories: theoretical solutions, simulation-based solutions, and benchmark-based solutions. However, these approaches often fail to simultaneously meet the requirements of generality, speed, and accuracy. Theoretical solutions calculate sketch configurations using simple mathematical formulas. For most sketches [11]–[15], theoretical configurations typically provide a bound in the form of $Pr\left\{\left|\hat{f}_i - f_i\right| \geq T\right\} \leq p$, where $\hat{f}_i$ is the estimated frequency of the flow, f_i is the actual frequency of the flow, T is an integer, and p is a probability. However, as noted in SketchConf [16], these bounds are often much looser than the real error for most workloads, resulting in inaccurate sketch configurations. Simulation-based solutions, such as SketchConf, employ Monte Carlo simulations to obtain accurate estimates of $Pr\left\{\left|\hat{f}_i - f_i\right| \geq T\right\} \leq p$. However, SketchConf is limited to frequency estimation and does not address configurations for other tasks. Furthermore, $Pr\left\{\left|\hat{f}_i - f_i\right| \geq T\right\} \leq p$ cannot solve for the AAE constraint in frequency estimation and cardinality estimation or the RR constraint in heavy-hitter detection, restricting their applicability. Benchmark-based solutions, like AutoSketch [17], use Latin Hypercube Sampling to sample configuration parameters and then conduct experimental testing to verify if they meet user-defined error constraints. While experimental testing is effective, it is quite time-consuming.

In summary, as presented in Table I, an ideal configuration framework should fulfill the following three key requirements:

- **General.** The framework should be applicable to a broad range of tasks and sketches.

- **Rapid.** The configuration process should minimize time overhead.
- **Accurate.** The generated configurations should closely align with real-world performance and meet user-defined error constraints.

This paper aims to address the aforementioned challenges by developing a configuration framework that is suitable for a wide range of tasks and most sketches. The proposed framework should rapidly and accurately generate sketch configurations that meet user-defined error constraints while minimizing memory overhead.

B. Proposed Solution and Contributions

In this paper, we propose RA-Sketch, a general framework that rapidly and accurately generates memory-optimal sketch configurations under the user-defined error constraints. Given multiple sketches performing different measurement tasks and specified error constraints for each sketch, RA-Sketch supports one-click generation of parameter configurations that meet user-defined error constraints while minimizing memory overhead. Specifically, our contributions are as follows:

Contribution I: rapid and accurate error predictor. The error predictor in RA-Sketch is based on two key lemmas to compute error metrics rapidly and accurately. For frequency-independent tasks, such as membership query and heavy-hitter detection, RA-Sketch utilizes Lemma-I to rapidly compute accurate FPR or RR for given parameters without requiring time-consuming experimental testing. For frequency-dependent tasks, such as frequency estimation and cardinality estimation, RA-Sketch employs Lemma-II. Monte Carlo simulations are used to estimate per-bucket collision effects, enabling rapid and accurate computation of AAE.

Contribution II: rapid hierarchical search strategy. RA-Sketch employs a three-step search strategy: an initial parameter initialization, a power-of-two scaling memory search, and a final binary search to refine parameter configurations that meet user-defined constraints. The well-designed initialization allows RA-Sketch to begin the search from a memory size close to meeting the constraints, eliminating the need to start binary search from the maximum memory size. This approach substantially reduces the number of search iterations, thereby enhancing search efficiency.

Contribution III: extensive experimental verification. To validate the effectiveness of RA-Sketch, we apply it to two real-world network traces and multiple sketches across four types of tasks. The experimental results demonstrate that generated configurations closely match the error metrics observed in real-world scenarios. Moreover, RA-Sketch achieves configuration speeds that are several orders of magnitude faster than baseline solution.

II. Related Work

A. Different Kinds of Sketches

1) Sketches for Membership Query: Membership query checks whether a flow is present. Due to its memory efficiency and fast query/update speed, the Bloom Filter [18] has been widely adopted. Recently, variants of Bloom Filter have been proposed to meet the requirements of different applications [19]–[22].

2) Sketches for Heavy-Hitter Detection: Heavy-hitter detection identifies flows that exceed a given frequency threshold. Existing approaches typically fall into two categories: min-heap-based approaches and preservation-based approaches. Min-heap-based solutions, such as the CM Sketch combined with a min-heap, use frequency-estimation sketches to maintain the top-k flows. Despite their simplicity, these solutions exhibit low processing speeds. Conversely, preservation-based solutions employ specialized algorithms to retain elephant flows and subsequently traverse the bucket array to detect flows surpassing the threshold. These solutions achieve superior processing speeds and recall rates, exemplified by Elastic Sketch [13], HeavyGuardian [14], HeavyKeeper [15], MV Sketch [23] and so on [24], [25].

3) Sketches for Frequency Estimation: Frequency estimation calculates the number of packets in a flow. Typical frequency estimation sketches include CM Sketch [11], CU Sketch [26] and CO Sketch [12]. Recent advancements exploit the skewed distribution of network traffic by differentiating between elephant flows and mouse flows, assigning counters with varying bit-lengths to enhance memory efficiency and accuracy. Representative examples include Tower Sketch [27], and BitSense [28].

4) Sketches for Cardinality Estimation: Each flow can be viewed as a pair $\langle f, e\rangle$. The cardinality of a flow is defined as the number of distinct e corresponding to f. gSkt [29] achieves memory-efficient multi-flow cardinality estimation by replacing the counters in CM Sketch with single-flow cardinality estimators. Building on the idea of CO Sketch, rSkt [30] effectively mitigates noise caused by hash collisions by additionally maintaining secondary cardinality estimators.

B. Prior Work on Sketch Configuration

Sketch configuration solutions are generally classified into three categories: theoretical solutions, simulation-based solutions, and benchmark-based solutions.

In some sketches supporting membership query, theoretical solutions can provide relatively accurate configurations, such as in Bloom Filter [18]. However, for most sketches [11]–[15] supporting frequency estimation, cardinality estimation and heavy-hitter detection, theoretical configurations only provide a bound in the form of $Pr\left\{\left|\hat{f}_i - f_i\right| \geq T\right\} \leq p$, where $\hat{f}_i$ is the estimated frequency of the flow, f_i is the actual frequency of the flow, T is an integer, and p is a probability. This is often much looser than the real error. More importantly, $Pr\left\{\left|\hat{f}_i - f_i\right| \geq T\right\} \leq p$ cannot solve for the AAE constraint in frequency estimation and cardinality estimation or the RR constraint in heavy-hitter detection.

The most representative simulation-based solution is SketchConf [16], which has been detailed in Section I-A and will not be reiterated here.

Benchmark-based solutions rely on experimental testing to determine whether user-defined error constraints are met. The

most advanced of these, AutoSketch [17], utilizes Latin Hypercube Sampling to efficiently sample configuration parameters and subsequently validates error constraints through experimental testing. While Latin Hypercube Sampling reduces the parameter search space, the experimental testing process requires inserting all packets and executing queries, making it computationally expensive. As a result, benchmark-based solutions remain substantially time-consuming.

In contrast to the aforementioned solutions, RA-Sketch does not apply a uniform configuration strategy to all sketches indiscriminately. Instead, it leverages the common hash distribution characteristics shared by all sketches and classifies them into frequency-dependent and frequency-independent sketches, adopting targeted strategies accordingly. Specifically, for frequency-dependent sketches, RA-Sketch resembles simulation-based solutions, using Monte Carlo simulation to quickly obtain the optimal configuration. However, unlike those solutions, it does not compute $Pr\left\{\left|\hat{f}_i - f_i\right| \geq T\right\} \leq p$. For frequency-independent sketches, RA-Sketch is similar to theoretical solutions, but instead of assuming completely uniform hashing, it utilizes hash distribution characteristics to obtain the optimal configuration.

III. The Design of RA-Sketch

A. Baseline and Our Observations

Most sketches have configurable parameters, including the number of hash functions h, the number of rows d, and the number of columns w. In most sketches, h and d are set to be equal [11], [12], [15], [29].

Baseline: The baseline solution performs a binary search on memory and then conducts experimental testing to verify whether the user-defined error constraints are met. For each memory size, it iterates over possible sketch row numbers d and their corresponding column numbers w. As previous practice has shown that setting d to 1–3 is generally sufficient for most sketches, we only need to iterate over lower values of d, such as $d = 1$–3.

Analysis: The baseline can find the memory-optimal configurations, but its time efficiency is poor due to binary search and experimental testing. This is because the experimental testing requires inserting all packets and performing queries to verify whether the user-defined error constraints are met, which is quite time-consuming. Additionally, the binary search starting from the maximum memory leads to many unnecessary search iterations. Therefore, we aim to replace experimental testing and restructure the search strategy to enhance the time efficiency of the configuration process.

Discussion: Why are theoretical or simulation-based solution not used as the baseline? The primary reason is that these solutions cannot address error constraints across diverse tasks and are often tailored to specific tasks or sketches.

B. Error Predictor of RA-Sketch

The error predictor is based on two key lemmas:

Lemma-I: When N flows are randomly mapped into w buckets, let Z be the number of flows in any given bucket. Then Z follows a binomial distribution with parameters N and $\frac{1}{w}$. If N is large, Z can be approximated by a Poisson distribution with $\lambda = \frac{N}{w}$.

Lemma-II: When N flows are randomly mapped into w buckets, for any flow f, the probability that any of the other $N-1$ different flows collides with f is $\frac{1}{w}$. Let Z be the number of distinct collision items, then Z follows a binomial distribution with parameters $N-1$ and $\frac{1}{w}$. If $N-1$ is large, Z can be approximated by a Poisson distribution with $\lambda = \frac{N-1}{w}$.

The probability mass function (PMF) of a Poisson distribution is defined as $P(Z=k) = \frac{\lambda^k e^{-\lambda}}{k!}$, where k is the number of occurrences of the event, λ is the average rate of occurrence, e is the base of the natural logarithm.

These two lemmas have been formally proven in previous works, such as SeqHash [31] and SketchConf [16]. It is worth noting that these two lemmas capture different aspects of the sketch behavior. Lemma-I reflects the number of flows contained in a bucket, while Lemma-II captures the number of collisions each flow has with other flows.

Insight: We classify existing sketches into two categories: frequency-dependent sketches and frequency-independent sketches. Frequency-dependent sketches are those whose error constraints are influenced by flow frequencies. For example, in the CM Sketch, the error of each flow is related to the frequencies of the other flows that hash to the same bucket. In contrast, frequency-independent sketches are those whose error constraints are independent of the flow frequencies. For example, in the Bloom Filter, the FPR depends only on the number of distinct flows, regardless of their individual frequencies. For frequency-independent tasks, such as membership query and heavy-hitter detection, RA-Sketch leverages Lemma-I to rapidly compute accurate FPR or RR for given parameters without relying on time-consuming experimental testing. For frequency-dependent tasks, such as frequency estimation and cardinality estimation, RA-Sketch employs Lemma-II and uses Monte Carlo simulation to estimate the per-bucket collision effects, enabling rapid and accurate computation of AAE. In cardinality estimation, frequency-dependent refers to cardinality-dependent and will not be further elaborated hereafter.

Discussion: Why should heavy-hitter detection be classified as a frequency-independent task? Although it lies between frequency-dependent and frequency-independent tasks, it is closer to the latter. This is because heavy-hitter detection algorithms are specifically designed to make elephant flows resistant to the influence of colliding flows that hash into the same bucket, thereby shielding elephant flows from interference. The more effective the algorithm is, the better it suppresses the influence of colliding flows, regardless of their frequencies. Therefore, we classify heavy-hitter detection as a frequency-independent task. This claim is further supported by the experimental results presented in Section V. Moreover, despite the presence of some estimation errors, the error remains within 10%, as shown in Section V.

We now consider how to determine whether the error constraints can be met given the frequency/cardinality distribution D, the number of distinct flows N, and the sketch configu-

Algorithm 1 Error Predictor for Frequency-Independent Task

Input: the number of distinct flows N; the number of hash functions h; the number of rows d; the number of columns/buckets per row w; the number of elephant flows H; frequency/cardinality distribution D.
Output: error.

1: **function** ERROR PREDICTOR FOR BLOOM FILTER(N,h,w)
2: // Z: the number of flows mapped to any given bit
3: $\lambda \leftarrow \frac{N \times h}{w}$
4: $Z \sim Poisson(\lambda)$
5: $P(Z = 0) \leftarrow e^{-\lambda}$
6: **return** $[1 - P(Z = 0)]^h$
7: **end function**
8: **function** ERROR PREDICTOR FOR ELASTIC SKETCH(H,w)
9: // Z: the number of elephant flows in any given bucket
10: $\lambda \leftarrow \frac{H}{w}$
11: $Z \sim Poisson(\lambda)$
12: $rr \leftarrow 0$, $temp\_pro \leftarrow 1 - e^{-\lambda}$
13: **for** $k = 1 \rightarrow 7$ **do**
14: $P(Z = k) \leftarrow \frac{\lambda^k e^{-\lambda}}{k!}$
15: $rr \leftarrow rr + k \times P(Z = k)$
16: $temp\_pro \leftarrow temp\_pro - P(Z = k)$
17: **end for**
18: $rr \leftarrow rr + 7 \times temp\_pro$
19: $rr \leftarrow \frac{rr}{\lambda}$
20: **return** rr
21: **end function**

rations (h, d, w). Below, we illustrate how to construct the error predictor based on the two lemmas, using a representative sketch from the four types of tasks.

1) Bloom Filter [18] for Membership Query

Structure: The Bloom Filter consists of h hash functions and a bit array of length w ($d = 1$). When a flow is inserted, the Bloom Filter maps it to h bits using the h hash functions and sets those bits to 1. When querying a flow, it checks the h mapped bits, and returns true only if all h bits are 1. Bloom Filter has one-sided errors, leading to false positives.

Error Predictor *(line 1-7)*: Bloom Filter falls under frequency-independent sketch. Therefore, we use Lemma-I to derive its error constraint. As shown in Alg. 1, based on Lemma-I, let Z be the number of flows mapped to any given bit. Since the Bloom Filter maps each flow to h bits, Z follows a Poisson distribution with $\lambda = \frac{N \times h}{w}$, where N is the number of distinct flows. According to the PMF of Poisson distribution, the proportion of bit still set to 0 after inserting N flows is $P(Z = 0) = e^{-\lambda}$. Therefore, the proportion of bit set to 1 in the Bloom Filter is $1 - P(Z = 0)$, and FPR is $[1 - P(Z = 0)]^h$. Excitingly, this matches the theoretical error formula of the Bloom Filter. It is well-known that when w is large, the theoretical error formula of the Bloom Filter closely matches its actual FPR.

2) Elastic Sketch [13] for Heavy-Hitter Detection

Structure: The Elastic Sketch consists of two parts: a heavy part for recording elephant flows and a light part for recording mouse flows. In the software version of the Elastic Sketch, the heavy part consists of a single bucket array ($d = h = 1$), with each bucket storing up to 7 flows and 1 $vote^-$ counter. And bucket array has a length of w. When a flow is inserted, if it maps to a cell in the heavy part that already contains the flow or if there is an empty cell, the Elastic sketch inserts it into the heavy part. Otherwise, the Elastic Sketch uses a voting mechanism to determine whether to insert the flow into the heavy or light part.

Error Predictor *(line 8-21)*: When addressing heavy-hitter detection, Elastic Sketch falls under frequency-independent sketch. Therefore, we use Lemma-I to derive its error constraint. As shown in Alg. 1, based on Lemma-I, let Z be the number of elephant flows in any given bucket. Assuming there are H elephant flows, Z follows a Poisson distribution with parameter $\lambda = \frac{H}{w}$. According to the PMF of Poisson distribution, we calculate the proportion of each possible value of Z and multiply it by the corresponding value to represent the number of identified elephant flows. Since each bucket in the heavy part of the Elastic Sketch can store up to 7 elephant flows, when $Z > 7$, we multiply its proportion by 7. Finally, we divide by λ to obtain the RR of the Elastic Sketch.

3) CM Sketch [11] for Frequency Estimation

Structure: The CM Sketch consists of d equal-length counter arrays and h hash functions ($d = h$). Each counter array has a length of w. When a flow is inserted, the CM Sketch maps it to d counters using the d hash functions, incrementing each counter by 1. When querying a flow, it checks the d mapped counters and reports the minimum value.

Error Predictor *(line 1-16)*: CM Sketch falls under frequency-dependent sketch. Therefore, we use Lemma-II to derive its error constraint. As shown in Alg. 2, based on Lemma-II, let Z be the number of distinct colliding flows, which follows a Poisson distribution with $\lambda = \frac{N-1}{w}$, where N is the number of distinct flows. We use a Monte Carlo simulation to calculate the AAE of the CM Sketch. For each counter, we repeatedly generate the number of collisions n according to the Poisson distribution. We then randomly sample n flows from the frequency distribution D to compute the total frequency. Since the CM Sketch inserts into all d arrays, we repeat this process d times. Since the CM Sketch reports the minimum value among the d mapped counters, we take the minimum value of the total frequencies as the estimation error. We repeat this process until the AAE converges.

4) rSkt1 [30] for Cardinality Estimation

Structure: The rSkt1 consists of d equal-length bucket arrays and h hash functions ($d = h$). Additionally, each array is associated with an independent auxiliary hash function $g_i()$ ($1 \leq i \leq d$). Each bucket array has a length of w, and each bucket contains a primary cardinality estimator and a secondary estimator. When inserting a flow, the rSkt1 maps it to d buckets using the d hash functions, and the corresponding $g_i()$ determines whether the flow updates the primary or secondary estimator in each bucket. When querying a flow,

Algorithm 2 Error Predictor for Frequency-Dependent Task

1: **function** ERROR PREDICTOR FOR CM SKETCH(N,d,w)
2: // Z: the number of distinct colliding flows
3: $\lambda \leftarrow \frac{N-1}{w}$
4: $Z \sim Poisson(\lambda)$
5: $aae \leftarrow 0$, $aae\_tot \leftarrow 0$, $num \leftarrow 0$
6: **while** aae has not converged **do**
7: $min\_sum \leftarrow (1 \ll 30)$
8: **for** $k = 1 \rightarrow d$ **do**
9: draw n from Z
10: $min\_sum \leftarrow min(min\_sum$, sum of n frequencies sampled from $D)$
11: **end for**
12: $aae\_tot \leftarrow aae\_tot + min\_sum$, $num \leftarrow num+1$
13: $aae \leftarrow \frac{aae\_tot}{num}$
14: **end while**
15: **return** aae
16: **end function**
17: **function** ERROR PREDICTOR FOR RSKT1(N,d,w)
18: // Z: the number of distinct colliding flows
19: $\lambda \leftarrow \frac{N-1}{w}$
20: $Z \sim Poisson(\lambda)$
21: $aae \leftarrow 0$, $aae\_tot \leftarrow 0$, $num \leftarrow 0$
22: **while** aae has not converged **do**
23: $min\_sum \leftarrow (1 \ll 30), aae\_temp \leftarrow 0$
24: **for** $k = 1 \rightarrow d$ **do**
25: $primary \leftarrow 0, secondary \leftarrow 0$
26: draw n from Z
27: **for** $j = 1 \rightarrow n$ **do**
28: $card \leftarrow$flow cardinality sampled from D
29: **if** $rand()\%2$ **then**
30: $primary \leftarrow primary + card$
31: **else**
32: $secondary \leftarrow secondary + card$
33: **end if**
34: **end for**
35: **if** $min\_sum > primary + secondary$ **then**
36: $min\_sum \leftarrow primary + secondary$
37: $aae\_temp \leftarrow abs(primary - secondary)$
38: **end if**
39: **end for**
40: $aae\_tot \leftarrow aae\_tot + aae\_temp$, $num \leftarrow num+1$
41: $aae \leftarrow \frac{aae\_tot}{num}$
42: **end while**
43: **return** aae
44: **end function**

the rSkt1 selects the bucket with the smallest total cardinality among the d mapped buckets and uses $g_i()$ to decide whether to compute the final estimate as the difference between the primary and secondary estimators or vice versa.

Error Predictor: *(line 17-44)*: rSkt1 falls under frequency-dependent sketch. Therefore, we use Lemma-II to derive its error constraint. As shown in Alg. 2, based on Lemma-II, let Z be the number of distinct colliding flows, which follows a Poisson distribution with $\lambda = \frac{N-1}{w}$, where N is the number of distinct flows. We use a Monte Carlo simulation to calculate the AAE of the rSkt1. For each bucket, we repeatedly generate the number of collisions n according to the Poisson distribution. We then randomly sample n flows from the cardinality distribution D and assign them to the primary and secondary estimators. The total cardinality of each estimator is then computed independently. Since the rSkt1 inserts into all d arrays, we repeat this process d times. Finally, rSkt1 selects the bucket with the minimum total cardinality among the d mapped buckets and computes the difference between the primary and secondary estimates as the final cardinality estimate. Therefore, we take the absolute difference between the two estimates as the estimation error. We repeat this process until the AAE converges.

Analysis: We demonstrate the construction of error predictors based on Lemma-I and II using four representative sketches. As shown in Section IV, a significant number of sketch error predictors can be constructed based on these two lemmas, which demonstrates the generality of RA-Sketch. Compared to theoretical solutions, RA-Sketch is not only more general, but also more accurate in predicting the actual sketch errors. Compared to simulation-based solutions, RA-Sketch is more general. Compared to benchmark-based solutions, RA-Sketch achieves faster configuration search by eliminating the need for time-consuming experimental testing.

C. Hierarchical Search Strategy of RA-Sketch

After preparing the error predictor, we need to design a search strategy to find the memory-optimal configurations. The reason why binary search is time-consuming is that it starts from the maximum memory and gradually bisects, resulting in many unnecessary searches. For example, suppose the maximum memory limit allocated to the sketch is 10 MB. In a Bloom Filter with $10,000$ flows, it is assumed that each flow is hashed only once. Now we need to determine the bit array length w to allocate to the Bloom Filter to meet FPR$\leq 10\%$. If using binary search, we would start from 10 MB, i.e., $w = 84$ million. However, even if each flow occupies a unique bit, the number of bits needed to meet FPR$\leq 10\%$ would not exceed $100,000$. Therefore, we propose starting the search from a reasonable initial length. Compared to start from the maximum memory, starting the search from $w = 100,000$ reduces 10 unnecessary searches.

Insight: RA-Sketch employs a three-step search strategy: an initial parameter initialization, a power-of-two scaling memory search, and a final binary search to refine parameter configurations that meet user defined constraints. The initialization of RA-Sketch is primarily based on a key insight: determining the number of buckets required by a sketch under the ideal scenario in which flows are uniformly hashed. Although perfect uniform hashing is unattainable in practice, this estimation typically serves as a lower bound on the required number of buckets. Due to the non-uniformity of practical hash functions [32], [33], more buckets are often needed. Moreover, this lower bound is closer to the actual requirement under real

Algorithm 3 Configuration Search for Bloom Filter

Input: the number of distinct flows N; error constraints FPR.

Output: the memory-optimal configurations.

function CONFIGURATION SEARCH FOR BLOOM FILTER(N,FPR)
  for $h = 1 \rightarrow 3$ **do**
    $w \leftarrow \frac{N \times h}{\sqrt[h]{FPR}}$
    $pre\_fpr \leftarrow$ Error Predictor(N,h,w)
    if pre_fpr is close to FPR **then** $res\_w \leftarrow w$
    else if $pre\_fpr < FPR$ **then**
      while $\frac{w}{2}$ **do**
        $w \leftarrow \frac{w}{2}$
        $pre\_fpr \leftarrow$ Error Predictor(N,h,w)
        if pre_fpr is close to FPR **then**
          $res\_w \leftarrow w$
          break
        else $pre\_fpr > FPR$
          $low \leftarrow w$, $high \leftarrow w \times 2$
          binary search to obtain res_w
          break
        end if
      end while
    else if $pre\_fpr > FPR$ **then**
      while $w \times 2$ **do**
        $w \leftarrow w \times 2$
        $pre\_fpr \leftarrow$ Error Predictor(N,h,w)
        if pre_fpr is close to FPR **then**
          $res\_w \leftarrow w$
          break
        else $pre\_fpr < FPR$
          $low \leftarrow \frac{w}{2}$, $high \leftarrow w$
          binary search to obtain res_w
          break
        end if
      end while
    end if
    $result.push(h, res\_w)$
  end for
  return memory-optimal configurations from $result$
end function

hash functions than the upper bound implied by the maximum memory size. Therefore, this initialization approach is more reasonable than performing a binary search starting from the maximum memory size. We use the four representative sketches from the previous section to illustrate how to perform reasonable initialization and describe our search strategy.

1) Bloom Filter for Membership Query

As shown in Alg. 3, for a given FPR and number of flows N, we initialize w as $\frac{N \times h}{\sqrt[h]{FPR}}$ for different numbers of hash functions h, which corresponds to the case where flows are evenly hashed. We then use the error predictor to estimate the current false positive rate pre_fpr. When $pre\_fpr > FPR$, the memory is doubled iteratively until $pre\_fpr < FPR$; when $pre\_fpr < FPR$, the memory is halved iteratively until $pre\_fpr > FPR$. Subsequently, we perform binary search to obtain the memory-optimal configurations for the current h. We repeat this process for each h. Finally, we select memory-optimal configurations of h and w.

Discussion: Although the formulas from previous Bloom Filter-related studies [18], [34] can generate optimal configurations, their direct application is not always feasible. For example, for 250,000 flows with a 0.1% FPR, the optimal configuration requires 449 KB of memory and $h = 10$. However, allocating 10 hash functions for Bloom Filter in programmable switches is unacceptable, and even in servers, larger values of h cannot be used due to throughput limitations. In RA-Sketch, h is a user-defined upper bound, and RA-Sketch determines the optimal configuration under this constraint.

Due to space limitations, we do not provide the pseudocode of the following sketches, but they are similar to Alg. 3.

2) Elastic Sketch for Heavy-Hitter Detection

For a given RR and number of heavy hitters H, we initialize w as $\frac{H \times RR}{7}$, which corresponds to the case where elephant flows are evenly hashed and flow sizes remain unchanged. We then use the error predictor to estimate the recall rate pre_rr. When $pre\_rr > RR$, the memory is halved iteratively until $pre\_rr < RR$; when $pre\_rr < RR$, the memory is doubled iteratively until $pre\_rr > RR$. Finally, we perform binary search to obtain the memory-optimal configurations of w.

3) CM Sketch for Frequency Estimation

For a given AAE, total number of packets P, and number of flows N, we initialize w as $\frac{N}{\frac{AAE}{\frac{P}{N}}+1}$, where $\frac{P}{N}$ represents the average flow size and $\frac{AAE}{\frac{P}{N}} + 1$ denotes how many flows share a bucket, thus determining the ideal number of buckets required under the assumption of equal flow sizes and uniform hashing. We then use the error predictor to estimate the current AAE pre_aae. When $pre\_aae > AAE$, memory is iteratively doubled until $pre\_aae < AAE$; when $pre\_aae < AAE$, memory is iteratively halved until $pre\_aae > AAE$. We perform binary search to obtain the memory-optimal configurations for the current d. We repeat this process for each d. Finally, we select memory-optimal configurations of d and w.

4) rSkt1 for Cardinality Estimation

It's similar to the search algorithm in the CM Sketch, but with the total number of packets P replaced by the total number of cardinality C.

Analysis: Our initialization strategy provides a closer estimate of the actual number of buckets required under real hash functions than the upper bound implied by the maximum memory size, thereby avoiding many unnecessary searches and significantly enhancing time efficiency. Furthermore, based on this initialization, we subsequently adopt a strategy of power-of-two scaling followed by binary search.

D. Parameter Configurations for Multiple Sketches

Problem: Given a fixed memory budget, multiple sketches performing different measurement tasks, and specified error

constraints for each sketch, how can the parameters of all sketches be configured to satisfy these error constraints?

Solution: RA-Sketch independently determines the memory-optimal parameter configurations for each sketch to meet its respective error constraints. It then aggregates these configurations to verify compliance with the overall memory budget. This approach is feasible because RA-Sketch ensures that the parameter configurations for each sketch is optimal with respect to its memory usage. If the combined memory usage of all sketches exceeds the budget, it implies that fulfilling all error constraints within the specified memory limit is not achievable.

E. Comparison with SketchConf [16] and AutoSketch [17]

SketchConf is limited to frequency estimation and does not address configurations for other tasks. Furthermore, SketchConf is designed to compute the accurate $Pr\left\{\left|\hat{f}_i - f_i\right| \geq T\right\} \leq p$, but this formula cannot solve for the AAE in frequency estimation and cardinality estimation or the RR in heavy-hitter detection. As a result, it cannot be compared with our solution in the experiments.

The key difference between AutoSketch and baseline lies in the search strategy. However, AutoSketch also relies on experimental testing for configuration. As shown in Section V-C, the time efficiency of experimental testing is extremely low.

In contrast, our proposed RA-Sketch not only adapts to error constraints across various tasks, but also achieves superior time efficiency through Poisson-distributed collision modeling. Even when traffic experiences significant variations over time, RA-Sketch provides the most accurate configuration and the shortest reconfiguration time compared to other solutions.

F. Summary

In summary, RA-Sketch addresses two limitations of the baseline. Firstly, by incorporating Lemma-I and II into the sketch configuration, we construct accurate and efficient error predictors for various sketches through Poisson-distributed collision modeling, thereby eliminating the time overhead from experimental testing. Secondly, by initializing memory appropriately and subsequently employing a strategy of power-of-two scaling followed by binary search, we circumvent numerous futile searches, substantially enhancing the time efficiency of the configuration process.

IV. Generalize to other sketches

A. Applying RA-Sketch to CO Sketch [12]

Structure: The CO Sketch shares a similar structure to the CM Sketch, with the key difference being the update process. When inserting a flow, the CO Sketch randomly increments or decrements the d counters mapped by the hash functions, rather than always incrementing them. When querying a flow, the median value of the d mapped counters is reported.

Error Predictor: The error predictor for the CO Sketch differs from the CM Sketch in two ways. First, the size of each colliding flow has a 50% probability of being negative. Second, the absolute value of the median, rather than the minimum, of the d mapped counters is used as the AAE.

Search Strategy: The search strategy for the CO Sketch is consistent with that of the CM Sketch.

B. Applying RA-Sketch to gSkt [29]

Structure: The difference from the CM Sketch is the replacement of counters with single-flow cardinality estimators.

Error Predictor: It's similar to the error predictor in the CM Sketch, but with the frequency distribution replaced by the cardinality distribution.

Search Strategy: The search strategy for the gSkt is consistent with that of the rSkt1.

C. Applying RA-Sketch to HeavyGuardian [14]

Structure: When handling heavy-hitter detection, the HeavyGuardian consists of a single bucket array ($d = h = 1$), with each bucket storing up to 8 flows. When inserting a flow, if it is already present in the bucket or there is an empty slot, the flow is inserted. Otherwise, the HeavyGuardian applies an exponential decay to the smallest flow in the bucket, replacing it with the new flow once the decayed value reaches 0.

Error Predictor: Based on Lemma-I, let Z be the number of elephant flows in any given bucket, which follows a Poisson distribution with parameter $\lambda = \frac{H}{w}$, where H is the total number of elephant flows, and w is the number of buckets. According to the PMF of Poisson distribution, we calculate the proportion of each possible value of Z and multiply it by the corresponding value to represent the number of identified elephant flows. Since each bucket in the heavy part can store up to 8 elephant flows, when $Z > 8$, we assume that collisions among elephant flows result in none being identified, and thus multiply the proportion by 7 instead of 8. Finally, we divide the result by λ to obtain the RR.

Search Strategy: For a given RR and number of heavy hitters H, we initialize w as $\frac{H \times RR}{8}$, which corresponds to the case where elephant flows are evenly hashed and flow sizes remain unchanged. The search strategy is consistent with that of the Elastic Sketch.

D. Applying RA-Sketch to HeavyKeeper [15]

Structure: HeavyKeeper consists of d equal-length bucket arrays and h hash functions ($d = h$). When inserting a flow, HeavyKeeper maintains the keys of the most frequent flows in d mapped buckets. For a new flow, HeavyKeeper decays the count values of the existing flows in the buckets with a certain probability. When the count value reaches 0, the new flow is recorded in the bucket. When querying a flow, it checks the d mapped buckets and reports the maximum value among them.

Error Predictor: Based on Lemma-I, let Z be the number of elephant flows in any given bucket, which follows a Poisson distribution with parameter $\lambda = \frac{H}{w}$, where H is the total number of elephant flows, and w is the number of buckets. Since each bucket can store at most one elephant flow, we continue to assume that collisions among elephant flows result in none being identified. We calculate the proportion of $P(Z = 1)$ to

represent the number of identified elephant flows. Then, we divide by λ to obtain the RR for each layer. Assuming the identification of elephant flows in each layer is independent, we calculate the RR for d layers using $1-(1-P(Z=1))^d$.

Search Strategy: For a given RR and number of heavy hitters H, we initialize w as $H \times \left(1-\sqrt[d]{1-RR}\right)$ for different values of d, which corresponds to the case where elephant flows are uniformly hashed and are treated equally. The search strategy is consistent with the Elastic Sketch, except that it iterates over different numbers of hash functions.

E. Discussion

1) RA-Sketch is also applicable to other sketches, such as OneSketch [24], WavingSketch [35], Tower Sketch [27], UA Sketch [36], rSkt2 [30], and so on [37]. However, due to space limitations, these sketches are not discussed further.

2) The error predictor can be adjusted using prior knowledge to better align with real-world error constraints. The configurations presented in this work represent one possible approach. For example, in the case of HeavyGuardian, the assumption that collisions between elephant flows result in none being identified can be reconsidered. Collisions may still allow the identification of an elephant flow, particularly if one flow's frequency substantially exceeds the combined frequencies of the colliding flows. While both scenarios are plausible, we adopt the former assumption for simplicity.

V. Experimental results

A. Test Setup

Datasets: Our evaluation utilizes two real-world datasets.

- **CAIDA Dataset:** The first is the CAIDA dataset [38], comprising real Internet traffic traces sourced from CAIDA. We extract 25 million consecutive packets, resulting in approximately $260,000$ flows when aggregated by source IP and about $920,000$ flows when aggregated by source–destination IP pairs. Under source IP aggregation, there are approximately $11,000$ elephant flows using a heavy-hitter threshold of 250.
- **MAWI Dataset:** The second is the MAWI dataset [39], comprising real Internet traffic traces collected by the MAWI Working Group of the WIDE Project. We extract 25 million consecutive packets, resulting in approximately $90,000$ flows when aggregated by source IP and about 2.3 million flows when aggregated by source–destination IP pairs. Under source IP aggregation, there are approximately $3,000$ elephant flows using a heavy-hitter threshold of 250.

Implementation: All code is implemented in C++. We adopt the Bob hash as recommended in [32], and observe similar results with other hash functions, such as MurmurHash3 [33] and CityHash [40]. All the programs run on a machine with an Intel Core i9-13900H processor, 2.6 GHz, 14 cores, 20 threads, and 64GB DDR4 memory. The source code is available at Github [41].

Abbreviations: We introduce some abbreviations used in the experiment, using the CM Sketch as an example:

- BS_CM: This solution uses the baseline to predict errors and search for the configurations.
- RA_CM: This solution uses the RA-Sketch to predict errors and search for the configurations.
- BS_CM_1 and RA_CM_1: The final digit represents the number of hash functions used.

B. Experiments on Configuration Accuracy

We evaluate the error predictor across four types of tasks and multiple sketches, demonstrating that the error predictor based on Lemma-I and II can accurately predict errors.

Settings: For the Monte Carlo simulations involved in frequency-dependent tasks, we conduct the simulations in batches, with each batch comprising 1000 iterations. The simulation stops and outputs the predicted error if the fluctuation in prediction error between the current and previous batches is less than 0.0001.

Membership Query: As shown in Fig. 1a and 2a, for the Bloom Filter, the error between our predicted FPR and the actual FPR is nearly always negligible as the number of hash functions and bits varies.

Heavy-Hitter Detection: As shown in Fig. 1b, 1c, 2b, and 2c, for Elastic Sketch and HeavyGuardian, the error between our predicted RR and actual RR is within 10% in most cases as the number of buckets changes. Similarly, as shown in Fig. 1d and 2d, for HeavyKeeper, as the number of hash functions and buckets varies, the error between our predicted RR and actual RR is also within 10% in most cases.

Frequency Estimation: As shown in Fig. 1e, 1f, 2e, and 2f, for the CM Sketch and CO Sketch, the error between our predicted AAE and the actual AAE is nearly always negligible as the number of hash functions and counters varies.

Cardinality Estimation: As shown in Fig. 1g, 1h, 2g, and 2h, for the gSkt and rSkt1, the error between our predicted AAE and the actual AAE is nearly always negligible as the number of hash functions and counters varies.

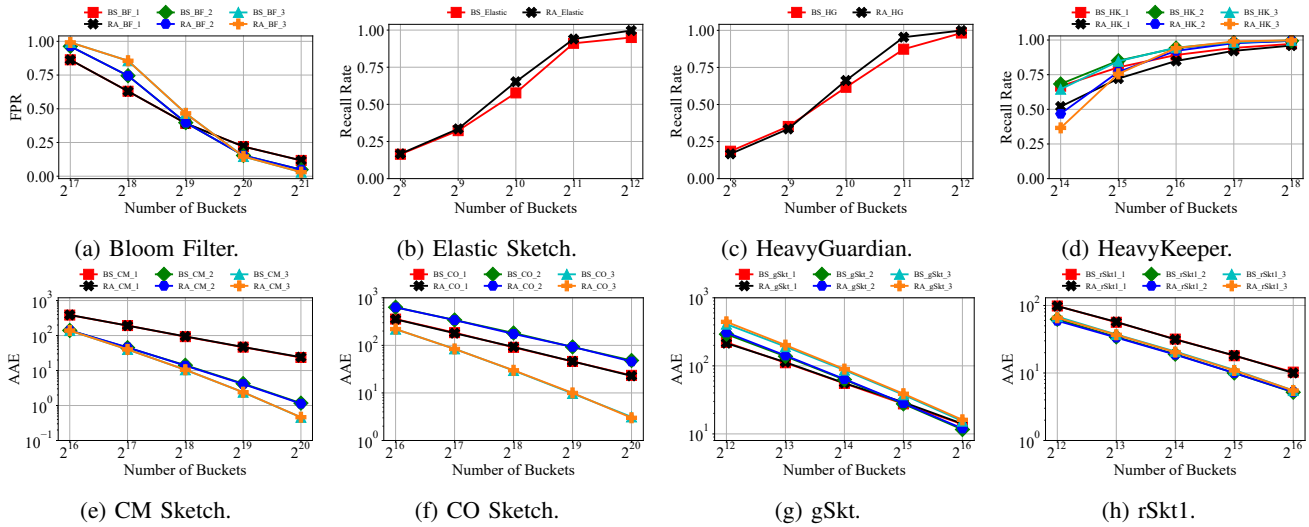
Analysis: The experimental results indicate that the error predictor of RA-Sketch can accurately predict error. For membership query, frequency estimation, and cardinality estimation, the predicted values are almost identical to the actual values. For heavy-hitter detection, the error between the predicted and actual values is within 10%.

Discussion: It is worth noting that applying RA-Sketch to low-performance heavy-hitter detection sketches results in a higher error. However, we think that the rationale for using low-performance sketches is insufficient when high-performance sketches, such as Elastic Sketch, are available.

C. Experiments on Configuration Search Time

We demonstrate the effectiveness of our search strategy using the number of iterations and evaluate the overall configuration speed of RA-Sketch through time cost.

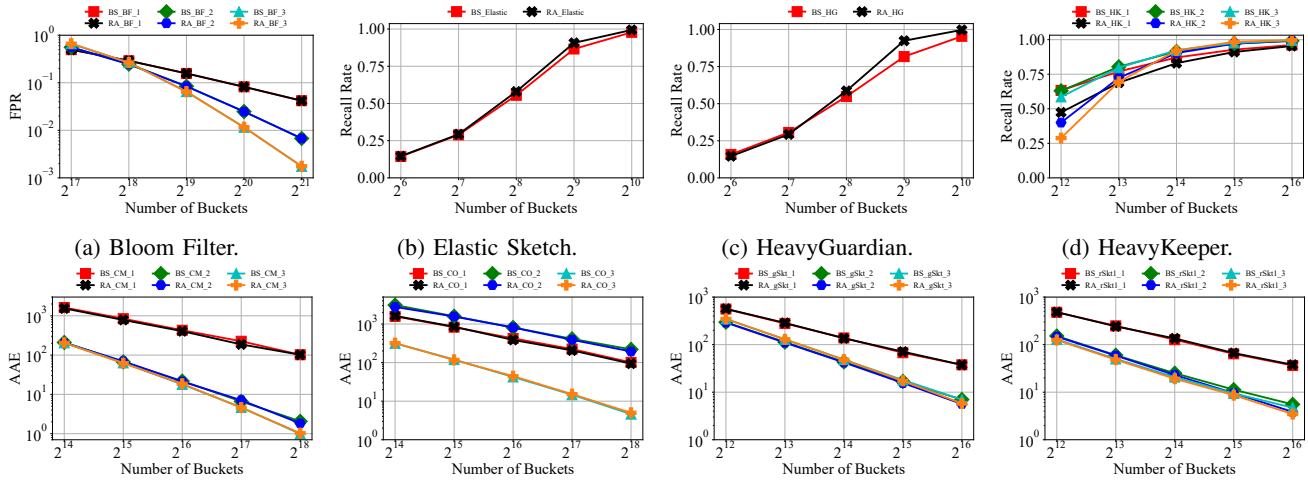
Settings: In the following experiments, the maximum memory allocation for the sketch is set to 10 MB, and the maximum number of hash functions h is limited to 3. The allowable error range in the algorithm is set to 5%, meaning the search stops



(a) Bloom Filter. (b) Elastic Sketch. (c) HeavyGuardian. (d) HeavyKeeper.

(e) CM Sketch. (f) CO Sketch. (g) gSkt. (h) rSkt1.

Fig. 1: Experiments on Configuration Accuracy of RA-Sketch using the CAIDA Dataset.



(a) Bloom Filter. (b) Elastic Sketch. (c) HeavyGuardian. (d) HeavyKeeper.

(e) CM Sketch. (f) CO Sketch. (g) gSkt. (h) rSkt1.

Fig. 2: Experiments on Configuration Accuracy of RA-Sketch using the MAWI Dataset.

when the predicted value deviates by 5% from the user-defined constraint. Except for the Bloom Filter, other sketches use the baseline mentioned in Section III-A for error prediction.

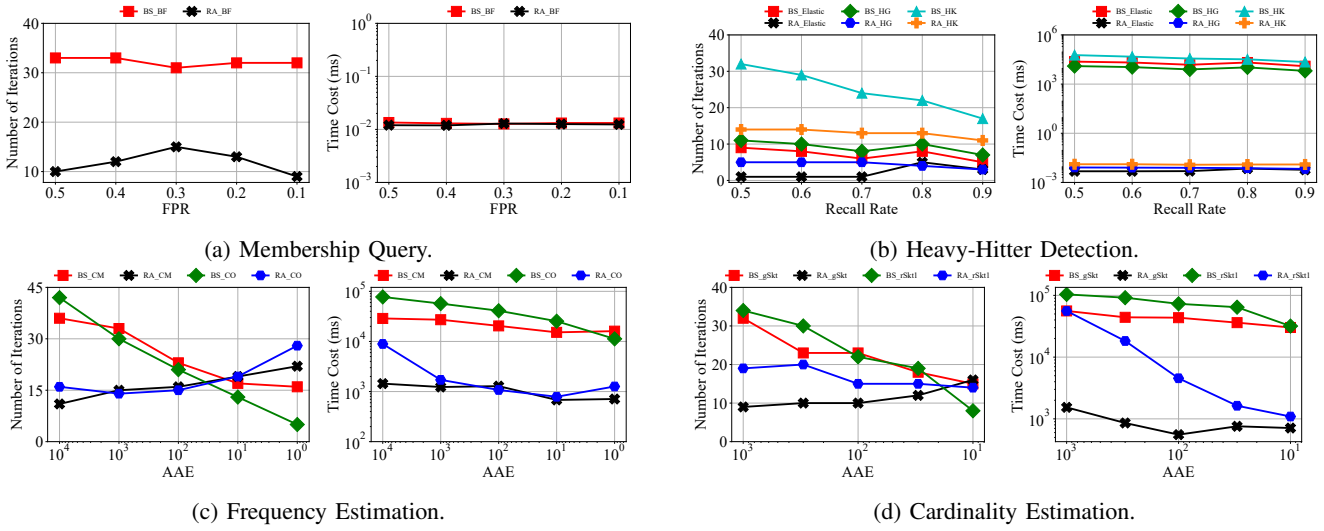
Membership Query: As shown in Fig. 3a and 4a, for the Bloom Filter, RA-Sketch reduces the average number of unnecessary searches by 20-30. Since the computational time overhead of the error prediction formula is negligible, RA-Sketch does not substantially enhance time efficiency.

Heavy-Hitter Detection: As shown in Fig. 3b and 4b, for Elastic Sketch, HeavyGuardian, and HeavyKeeper, RA-Sketch reduces the average number of unnecessary searches by 5, 5 and 12, respectively, on the CAIDA dataset, and by 8, 7 and 16, respectively, on the MAWI dataset, compared to the baseline. Due to the time efficiency of the error predictor, the search time for Elastic Sketch, HeavyGuardian, and HeavyKeeper is reduced by 6-7 orders of magnitude.

Frequency Estimation: As shown in Fig. 3c and 4c, compared with the baseline, RA-Sketch reduces the average number of unnecessary searches for CM Sketch and CO Sketch by 8 and 4, respectively, on the CAIDA dataset, and reduces that of CM Sketch by 6 on the MAWI dataset. Due to time efficiency of the error predictor, the search time for CM Sketch and CO Sketch is reduced by 1-2 orders of magnitude.

Cardinality Estimation: As shown in Fig. 3d and 4d, for gSkt and rSkt1, RA-Sketch reduces the average number of unnecessary searches by 11 and 6, respectively, on the CAIDA dataset, and by 7 and 4, respectively, on the MAWI dataset, compared to the baseline. Due to the time efficiency of the error predictor, the search time for gSkt and rSkt1 is reduced by 1-2 orders of magnitude.

Analysis: The results indicate that RA-Sketch substantially enhances configuration speed. This improvement is primarily due to two factors: the efficiency of RA-Sketch's search strategy, which minimizes unnecessary searches, and the time



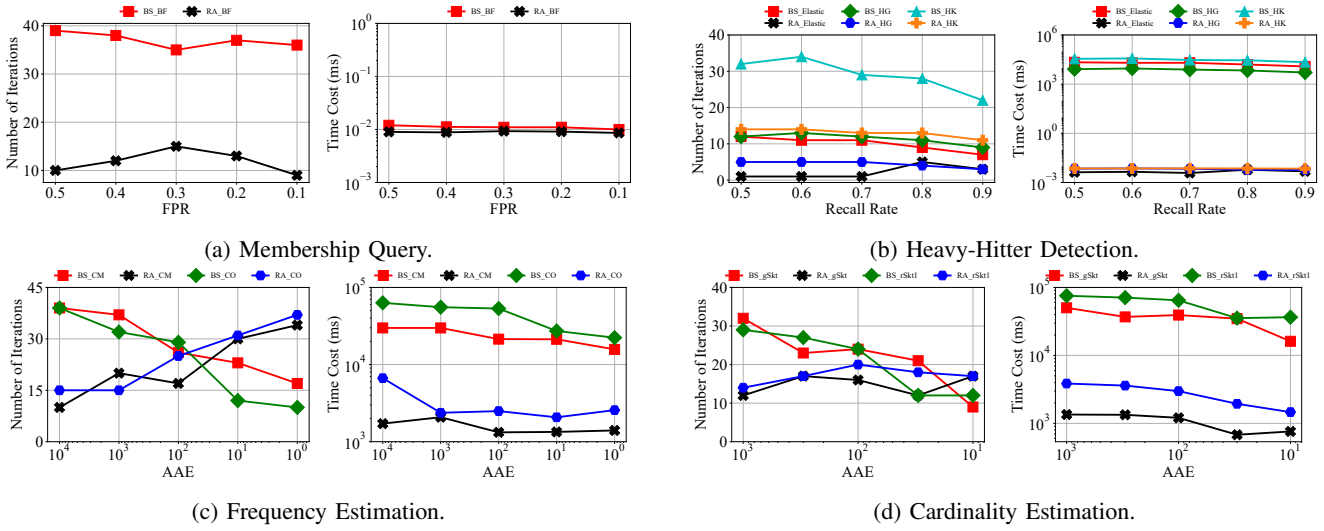
(a) Membership Query.

(b) Heavy-Hitter Detection.

(c) Frequency Estimation.

(d) Cardinality Estimation.

Fig. 3: Experiments on Configuration Search Time of RA-Sketch using the CAIDA Dataset.



(a) Membership Query.

(b) Heavy-Hitter Detection.

(c) Frequency Estimation.

(d) Cardinality Estimation.

Fig. 4: Experiments on Configuration Search Time of RA-Sketch using the MAWI Dataset.

efficiency of its error predictor, which eliminates the need for time-consuming experimental testing.

Discussion: It is worth noting that the configuration time of the baseline depends on data volume. For example, processing 25 million packets requires several tens of seconds for configuration, whereas processing 250 million packets takes several hundred seconds, and so on. In contrast, due to the Poisson-distributed collision modeling, the configuration time of RA-Sketch depends solely on the characteristics of the data distribution and is almost unaffected by the data volume.

VI. Conclusion

Providing parameter configurations that meet user-defined error constraints is critical for sketch applications across diverse scenarios. This paper presents RA-Sketch, a general framework that rapidly and accurately generates memory-optimal sketch configurations. RA-Sketch employs two key lemmas to construct accurate and rapid error predictors for various sketches, eliminating the need for time-consuming experimental testing. Furthermore, RA-Sketch introduces a hierarchical search technique with initialization, substantially reducing unnecessary searches. Experimental results demonstrate that RA-Sketch generates accurate configurations while substantially reducing configuration search time compared to benchmark-based solution.

Acknowledgments

We would like to thank our shepherd and the anonymous reviewers for their thoughtful feedback. This work is supported by the National Natural Science Foundation of China under Grant Nos. 62432003, U22B2005 and 62032013; the Liaoning Revitalization Talents Program under Grant No. XLYC2403086; and the financial support of Lingnan University (LU) under Grant No. DB23A9.

REFERENCES

[1] H. Zheng, C. Huang, X. Han, J. Zheng, X. Wang, C. Tian, W. Dou, and G. Chen, "μmon: Empowering microsecond-level network monitoring with wavelets," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 274–290.

[2] H. Zheng, C. Tian, T. Yang, H. Lin, C. Liu, Z. Zhang, W. Dou, and G. Chen, "Flymon: enabling on-the-fly task reconfiguration for network measurement," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 486–502.

[3] K. Yang, Y. Wu, R. Miao, T. Yang, Z. Liu, Z. Xu, R. Qiu, Y. Zhao, H. Lv, Z. Ji *et al.*, "Chamelemon: Shifting measurement attention as network state changes," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 881–903.

[4] Q. Huang, H. Sun, P. P. Lee, W. Bai, F. Zhu, and Y. Bao, "Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 404–421.

[5] L. Tang, Q. Huang, and P. P. Lee, "Spreadsketch: Toward invertible and network-wide detection of superspreaders," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 1608–1617.

[6] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, "Hpcc: High precision congestion control," in *Proceedings of the ACM special interest group on data communication*, 2019, pp. 44–58.

[7] Q. Huang, X. Jin, P. P. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 113–126.

[8] Y. Liu, K. Guo, F. Li, J. Shen, and X. Wang, "La-sketch: An adaptive level-aware sketch for efficient network traffic measurement," in *2025 IEEE/ACM 33nd International Symposium on Quality of Service (IWQoS)*. IEEE, 2025, pp. 1–10.

[9] K. Guo, F. Li, J. Shen, X. Wang, and J. Cao, "Distributed sketch deployment for software switches," *IEEE Transactions on Computers*, 2024.

[10] K. Guo, F. Li, J. Shen, and X. Wang, "Advancing sketch-based network measurement: A general, fine-grained, bit-adaptive sliding window framework," in *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. IEEE, 2024, pp. 1–10.

[11] 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.

[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. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 561–575.

[14] T. Yang, J. Gong, H. Zhang, L. Zou, L. Shi, and X. Li, "Heavyguardian: Separate and guard hot items in data streams," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2584–2593.

[15] T. Yang, H. Zhang, J. Li, J. Gong, S. Uhlig, S. Chen, and X. Li, "Heavykeeper: an accurate algorithm for finding top-k elephant flows," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 1845–1858, 2019.

[16] R. Miao, F. Dong, Y. Zhao, Y. Zhao, Y. Wu, K. Yang, T. Yang, and B. Cui, "Sketchconf: A framework for automatic sketch configuration," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 2022–2035.

[17] H. Sun, Q. Huang, J. Sun, W. Wang, J. Li, F. Li, Y. Bao, X. Yao, and G. Zhang, "{AutoSketch}: Automatic {Sketch-Oriented} compiler for query-driven network telemetry," in *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, 2024, pp. 1551–1572.

[18] 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.

[19] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM transactions on networking*, vol. 8, no. 3, pp. 281–293, 2000.

[20] Y. Wu, J. He, S. Yan, J. Wu, T. Yang, O. Ruas, G. Zhang, and B. Cui, "Elastic bloom filter: deletable and expandable filter using elastic fingerprints," *IEEE Transactions on Computers*, vol. 71, no. 4, pp. 984–991, 2021.

[21] H. Dai, J. Yu, M. Li, W. Wang, A. X. Liu, J. Ma, L. Qi, and G. Chen, "Bloom filter with noisy coding framework for multi-set membership testing," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[22] M. Li, R. Xie, D. Chen, H. Dai, R. Gu, H. Huang, W. Dou, and G. Chen, "A pareto optimal bloom filter family with hash adaptivity," *The VLDB Journal*, vol. 32, no. 3, pp. 525–548, 2023.

[23] L. Tang, Q. Huang, and P. P. Lee, "Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2026–2034.

[24] Z. Fan, R. Wang, Y. Cai, R. Zhang, T. Yang, Y. Wu, B. Cui, and S. Uhlig, "Onesketch: A generic and accurate sketch for data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 887–12 901, 2023.

[25] Y. Zhao, W. Zhou, W. Han, Z. Zhong, Y. Zhang, X. Zheng, T. Yang, and B. Cui, "Achieving top-k-fairness for finding global top-k frequent items," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[26] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, 2002, pp. 323–336.

[27] Y. Zhao, K. Yang, Z. Liu, T. Yang, L. Chen, S. Liu, N. Zheng, R. Wang, H. Wu, Y. Wang *et al.*, "Lightguardian: A full-visibility, lightweight, in-band telemetry system using sketchlets," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 991–1010.

[28] R. Ding, S. Yang, X. Chen, and Q. Huang, "Bitsense: Universal and nearly zero-error optimization for sketch counters with compressive sensing," in *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023, pp. 220–238.

[29] Y. Zhou, Y. Zhang, C. Ma, S. Chen, and O. O. Odegbile, "Generalized sketch families for network traffic measurement," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 3, no. 3, pp. 1–34, 2019.

[30] H. Wang, C. Ma, O. O. Odegbile, S. Chen, and J.-K. Peir, "Randomized error removal for online spread estimation in data streaming," *Proceedings of the VLDB Endowment*, vol. 14, no. 6, 2021.

[31] T. Bu, J. Cao, A. Chen, and P. P. Lee, "Sequential hashing: A flexible approach for unveiling significant patterns in high speed networks," *Computer Networks*, vol. 54, no. 18, pp. 3309–3326, 2010.

[32] C. Henke, C. Schmoll, and T. Zseby, "Empirical evaluation of hash functions for multipoint measurements," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 3, pp. 39–50, 2008.

[33] https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp.

[34] M. Mitzenmacher, "Compressed bloom filters," in *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, 2001, pp. 144–150.

[35] J. Li, Z. Li, Y. Xu, S. Jiang, T. Yang, B. Cui, Y. Dai, and G. Zhang, "Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1574–1584.

[36] J. Ye, L. Li, W. Zhang, G. Chen, Y. Shan, Y. Li, W. Li, and J. Huang, "Ua-sketch: an accurate approach to detect heavy flow based on uninterrupted arrival," in *Proceedings of the 51st International Conference on Parallel Processing*, 2022, pp. 1–11.

[37] H. Wang, "Enhancing accuracy for super spreader identification in high-speed data streams," *Proceedings of the VLDB Endowment*, vol. 17, no. 11, pp. 3124–3137, 2024.

[38] https://catalog.caida.org/dataset/passive_2018_pcap, Anonymized Internet Traces 2018.

[39] https://mawi.wide.ad.jp/mawi/samplepoint-F/2021/202109011400.html.

[40] https://github.com/aappleby/smhasher/blob/master/src/City.cpp.

[41] https://github.com/QingYeyyds/RA-Sketch.