OneSketch: A Generic and Accurate Sketch for Data Streams

Zhuochen Fan, Ruixin Wang, Yalun Cai, Ruwen Zhang, Tong Yang, *Member, IEEE*, Yuhan Wu, Bin Cui, *Senior Member, IEEE*, and Steve Uhlig

Abstract*—In this paper, we propose a generic sketch algorithm capable of achieving more accuracy in the following five tasks: finding top-k frequent items, finding heavy hitters, per-item frequency estimation, and heavy changes in the time and spatial dimension. The state-of-the-art (SOTA) sketch solution for multiple measurement tasks is ElasticSketch (ES). However, the accuracy of its frequency estimation has room for improvement. The reason for this is that ES suffers from overestimation errors in the light part, which introduces errors when querying both frequent and infrequent items. To address these problems, we propose a generic sketch, OneSketch, designed to minimize overestimation errors. To achieve the design goal, we propose four key techniques, which embrace hash collisions and minimize possible errors by handling highly recurrent item replacements well. Experimental results show that OneSketch clearly outperforms 12 SOTA schemes. For example, compared with ES, OneSketch achieves more than 10× lower Average Absolute Error on finding top-k frequent items and heavy hitters, as well as 48.3% and 38.4% higher F1 Scores on two heavy changes under 200KB memory, respectively.

***Index Terms*—Data streams, sketch, frequency estimation, top-k, heavy hitters, per-item, heavy changes**

I. Introduction

A. Background and Motivation

Approximate stream processing has always been a popular topic in various areas such as databases [1]–[4], data mining [5]–[7], artificial intelligence [8]–[11], network measurement [12]–[14] and security [15]–[17]. One of its most important and fundamental tasks is frequency estimation, which aims to accurately estimate the number of occurrences of a given item in data streams. Frequency estimation can generally be divided into estimation for frequent-item and per-item. Among them, frequent-item estimation can be further summarized as finding top-k items (**Task 1**) and heavy hitters (**Task 2**). The former finds the k items with the largest frequency, and the latter finds items whose frequency exceeds a predefined threshold. Per-item estimation (**Task 3**) on the other hand focuses on estimating the frequencies of all items. Two other tasks, the time dimension heavy changes and the spatial dimension heavy changes, are also important and are related to frequency estimation. Heavy changes in the time dimension (**Task 4**) refers to items whose frequencies in two adjacent time windows increase/decrease beyond a predefined threshold. Heavy changes in the spatial dimension (**Task 5**) is a new problem that we define for the first time, which refers to items whose frequencies in two adjacent physical nodes increase/decrease beyond a predefined threshold. It can be used for packet loss detection in networks, but has rarely been formally studied. Sketch, a compact data structure with small memory footprint and error, has been widely recognized by the research community [18]–[21], especially in addressing the above tasks 1 to 4. Thus, our design goal is to propose a generic sketch algorithm that can more accurately perform the above five tasks.

This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2020B0101390001, in part by the National Natural Science Foundation of China (NSFC) under Grant U20A20179. *(Zhuochen Fan, Ruixin Wang, and Yalun Cai are co-primary authors.) (Corresponding author: Tong Yang).*

Zhuochen Fan, Ruixin Wang, Yalun Cai, Ruwen Zhang, Yuhan Wu and Bin Cui are with the National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, Beijing 100871, China. (e-mail: $\{fanzc, caiyalun, zrw, yuhan.wu, bin.cui\}$@pku.edu.cn; wang.ruixin@stu.pku.edu.cn).

Tong Yang is with the National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, Beijing 100871, China, and also with the Peng Cheng Laboratory, Shenzhen 518066, China. (e-mail: yangtongemail@gmail.com).

Steve Uhlig is with the School of Electronic Engineering and Computer Science, Queen Mary University of London, London E1 4NS, U.K. (e-mail: steve@eecs.qmul.ac.uk).

B. Prior Art and Limitations

The distribution of data streams is highly skewed [22]–[24], $i.e.$, only a few items appear frequently (called frequent items), while most items appear only once or a few times (called infrequent items). Thus, researchers naturally pay more attention to frequent items and put forward many excellent works. The key idea of the most state-of-the-art (SOTA) sketch-based solutions, such as ElasticSketch (ES) [14], [25], MV-Sketch [26], Cold filter (CF) [27] and its successor LogLog Filter (LLF) [28], $etc.$, is to separate frequent items from infrequent items, and accurately record frequent items. Among them, ES is the most compelling: it handles multiple measurement tasks, including tasks 1 to 4, and offers a high level of accuracy.

While ES performs well, it has two obvious problems that cause inaccurate recording of frequent and infrequent items. ES has a heavy part and a light part, which record the frequencies of frequent and infrequent items, respectively. The heavy part records the support votes and negative votes for each item, representing the frequency of the item and the frequency of other items ($i.e.$, caused by hash collisions), respectively. Once a new item arrives and the ratio of negative votes exceeds a threshold, the original item is expelled to the light part, $i.e.$, an item replacement occurs. When querying the item frequency, for any item that is not in the heavy part, its frequency is reported by the light part. For any item in the heavy part, if item replacement has occurred in the past, the sum of the frequencies of two parts is reported. However, since the frequency of frequent items recorded in the light part is likely to introduce errors due to hash collisions, the final reported frequency must also have errors. Further, coupled with the fact that the light part uses a large-size counter to record infrequent items, it leads to a large memory overhead and serious overestimation errors in the light part.

C. Our Proposed Solution

To achieve higher accuracy, we propose a novel sketch algorithm called OneSketch, whose name ONE is inspired by two aspects: the measurement of five tasks can be realized using just one sketch, and the design philosophy is one-sided approaching (explain later). The accuracy of OneSketch in these five tasks is better than that of ES and other SOTA schemes: 1) For Task 1, the Average Relative Error (ARE) of OneSketch is on average 10.9× and 56.0× lower than that of ES and LLF, respectively. 2) For Task 2, the Average Relative Error (ARE) of OneSketch is on average 5.3× and 17.4× lower than that of ES and LLF, respectively. 3) For Task 3, OneSketch achieves 36.0% and 82.3% higher F1 Score than that of ES and LLF under 200KB of memory, respectively. 4) For Task 4, OneSketch achieves 48.3% and 72.4% higher F1 Score than that of ES and LLF under 200KB of memory, respectively. 5) For Task 5, OneSketch achieves 38.4% and 67.9% higher F1 Score than that of ES and LLF under 200KB of memory, respectively.

OneSketch inherits the ES idea of separating frequent and infrequent items, and its data structure also consists of two parts, a heavy part and a light part, which are used to accurately record the frequencies of frequent items and other infrequent items, respectively. The key design philosophy of OneSketch is **Overestimation Control**: reduce the overestimated frequency of items, by approaching the true frequency on one side. We propose four techniques around the above design philosophy that embrace the reality of hash collisions and minimize overestimation errors in terms of extremely recurrent item replacements.

The key technique for the light part is called **Fine-Grained Control**, which replaces the light part of ES from CM [29] to a tailored TowerSketch [30], exploiting its small-size counters to record infrequent items at a finer granularity and reduce memory overhead. This technique optimizes the accuracy of ES light part from the data structure.

The key technique of interaction between the light part and the heavy part is called **Frequency Read/Write Control** and **Repeat Control**. 1) The key idea of the former is that after each item replacement in the heavy part, the new challenger item that succeeds in the item replacement should immediately `read` its frequency recorded in the light part. The goal is to avoid possible overestimation errors in the light part due to hash collisions in the future. This is the main strength of OneSketch to address the overestimation errors generated by ES mentioned in Section I-B. Similarly, the light part should also `write` the frequency of the kicked item. 2) Whenever Frequency Read/Write Control occurs, the item frequency is recorded in the heavy/light part, but this process is reversible, $i.e.$, an item may be transferred repeatedly between heavy and light parts. As a result, if the process of Frequency Read/Write Control happens to an item many times, its estimated value in the light part will be accumulated repeatedly. Thus, our Repeat Control reduces the overestimation errors caused by above-mentioned issue, to further optimize Frequency Read/Write Control again.

The key technique for the heavy part is called **Replacement Control**, and the main idea is that we should control meaningless item replacement by comparing the minimum frequency of the mapped bucket in the heavy part with the read value of the new challenger item in the light part. This conservative technique eliminates the errors caused by the possible overflow of infrequent items in the light part, to further improve Frequency Read/Write Control.

The above techniques significantly reduce the overestimation errors of ES. More details are provided in Section III. Further, we develop a rigorous mathematical analysis for OneSketch to theoretically derive its error bounds in Section IV. Finally, we conduct extensive experiments, comparing OneSketch with 12 sketch-based SOTA schemes in Section V to verify its effectiveness. The experimental results show that OneSketch enables more accurate measurements in five important tasks in data streams. All related codes of OneSketch are provided open-source and available at GitHub [31][1].

Main Contributions:

1) We propose a new measurement task called heavy changes in the spatial dimension, which has important applications but has rarely been studied.
2) We propose OneSketch, which is generic for five tasks and more accurate than other SOTA solutions.
3) We theoretically derive the error bound for OneSketch through rigorous mathematical analysis.
4) We perform extensive experiments, and the results validate that OneSketch is generic and more accurate.

II. Related Work

In this section, we divide sketches for frequency estimation into *per-item estimation* and *frequent-item estimation*.

A. Per-Item Estimation

These sketches are designed to record the frequencies of all items. Typical algorithms include Count-Min sketch (CM) [29] and Conservative Update sketch (CU) [32]. A CM consists of d arrays $A_i (1 \leq i \leq d)$, where A_i is associated with a hash function $h_i(.)$, and each array has w counters. When inserting an item e, it increments the d hashed counter $A_i[h_i(e)]$ by 1. To report the frequency of e, it only reports the smallest one among the d hashed counters $A_i[h_i(e)]$. The CU is very similar to the CM, except that it only increments the minimum counter(s) among the d hashed counters for each insertion. They both suffer from overestimation errors due to hash collisions. Other well-known schemes include sketches of Count (C) [33] and CSM [34].

[1]https://github.com/pkufzc/OneSketch

SALSA [20] first uses small counters and accurately indicates the merging of adjacent counters when they overflow by complex operations with an additional bitmap, achieving high accuracy but sacrificing speed. It can be extended in CM, CU, and C versions. FCM-Sketch (FCMS) [35] and TowerSketch [30] both consist of several counter arrays. They use different-sized counters for different arrays, but each array is allocated the same amount of memory. Hence, the higher-level arrays have fewer counters, but their counters are larger. In this way, frequent items overflow in lower-level counters, so their frequencies are kept in higher-level/large counters, whereas the frequencies of infrequent items are kept in lower-level/small counters. FCMS tries its best to avoid counter overflow, and must rely on the existing schemes ($e.g.$, ES [14]) to achieve high accuracy in finding top-k frequent items. TowerSketch handles overflow well and can achieve high accuracy without requiring the cooperation of existing SOTA scheme. TowerSketch supports both CM and CU insertion, and we utilize the CU version of TowerSketch (denoted as Tower_CU) and redesign its insertion strategy in this paper.

B. Frequent-Item Estimation

Typical sketches include Misra-Gries sketch (MG) [36], Space-Saving (SS) [37], Unbiased Space-Saving (USS) [38], and ASketch (AS) [39]. As a pioneering work, when a new/non-recorded item arrives but the data structure is full, MG directly decrements the frequency of all recorded items by 1. It inspires many works such as SS and FD [40], but its replacement strategy will lose a lot of infrequent items and lead to a very low recall rate. SS and USS both use a data structure named Stream-Summary to record frequent items. Unlike MG, SS directly replaces the least frequent item with this new item, while USS utilizes probabilistic replacement to achieve unbiased estimation. AS uses a small array to record only a few frequent items and a sketch ($e.g.$, CM, C, FCM [41]) to record infrequent items, without guaranteeing the actual demand in terms of the total number of frequent items or the processing speed.

SOTA sketches include UnivMon (UM) [42], ElasticSketch (ES) [14], [25], MV-Sketch (MV) [26], [43], Cold filter (CF) [27] and its successor LogLog Filter (LLF) [28], $etc.$ UM is the first universal sketch to address multiple measurement tasks with a single data structure, based on the key idea of universal streaming [44] It first recursively samples the data stream to obtain several sub-streams, and uses C and heap to record each sub-stream. However, its actual accuracy is not satisfactory enough, and its sampling results in slow processing speed.

ES separates frequent items from infrequent items through a voting expulsion mechanism. ES consists of two parts: a heavy part records frequent items, and a light part records infrequent items. The heavy part is a hash table, where each bucket records the following item information: item ID, support vote, negative vote, and flag. Support vote records the frequency of this item. Negative vote records the frequency of other items. The flag indicates whether the light part is likely to contain support votes for this item, where the light part is a CM. When inserting an new item, if it differs from the item in the mapped bucket, it increments the negative vote, and calculates whether the ratio of the negative votes exceeds a predefined threshold. If so, the recorded item in the bucket is evicted to the light part. When querying an item, for any item that is not in the heavy part, its frequency is reported by the light part. For any item in the heavy part, there are two cases: 1) if the flag is false, then its frequency is the corresponding support vote; 2) if the flag is true, then its frequency is the sum of support vote and query result in the light part.

CF first uses a two-layer CU to record the frequency of all items, and then sets a predefined threshold to separate frequent items from infrequent items. When inserting an item, CF first inserts it into the CU and queries its frequency. If the frequency exceeds the threshold, the item will be reported as a frequent item. To expand the filtering range of CF, LLF replaces CU with the LogLog structure [45]–[47], which is originally used for cardinality ($i.e.$, the number of distinct items) estimation. Its data structure is an array of registers associated with a random generator and several hash functions. When inserting an item, LLF first calculates the hash functions to map the corresponding register, and determines whether it is an infrequent item. If the answer is positive, LLF generates random numbers, which follow a geometric distribution, and then updates the associated registers.

The structure of MV-Sketch (MV) [26], [43] is similar to that of CM. The three fields recorded in each bucket are: the sum of all item hashed to this bucket, the heavy (our Task 2&4) item candidates (candidates for short) in the current bucket, and the count value of the candidates in the current bucket. When an item is mapped to a bucket, MV uses MJRTY algorithm [48] to update the candidate. When querying an item, MV determines the estimated value based on whether the new item is consistent with the candidate, and returns the minimum estimated value. Finally, MV reports heavy items based on whether they are greater than the set threshold.

III. OneSketch Design

TABLE I: Symbols frequently used in this paper.

Notation	Meaning
e	A distinct item in the data stream
$\mathcal{B}[i][j]$	The j^{th} cell of the i^{th} bucket in the heavy part
d	Number of cells per bucket in the heavy part
$h(.)$	Hash function of the heavy part
C_H	Count value of item recorded in the heavy part
$\langle ID, C_H \rangle$	The two fields of the item recorded in the heavy part
$\mathcal{B}[h(e)]$	The hashed/mapped bucket of e in the heavy part
C'_H	Count value of the least frequent item e' in $\mathcal{B}[h(e)]$
$g_s(.)$	s^{th} hash function of the light part
$\mathcal{A}[s][g_s(e)]$	Hashed/mapped counters of e in the light part
C_L	Query value of the item in the light part
δ	the number of bits in the light part counter

In this section, we first present the data structure of OneSketch in Section III-A. Then, we introduce the design philosophy and four techniques of OneSketch in Section III-B. Next, we introduce the operations of the Light Part in Section III-C, as a prerequisite for the subsequent OneSketch operations. Finally, we describe the specific operations of OneSketch for

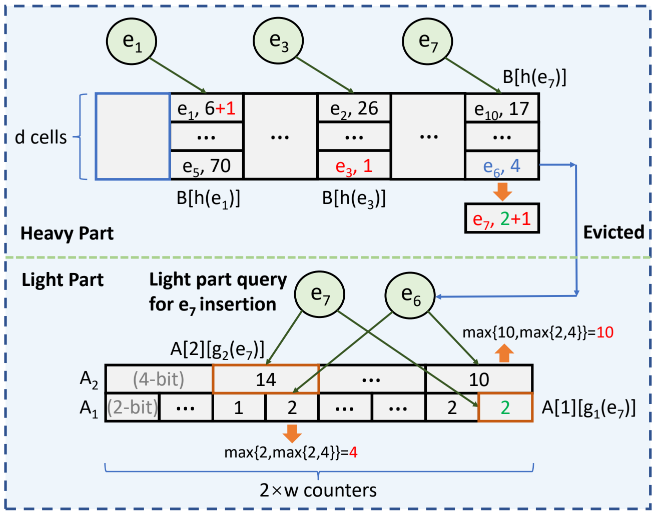


Fig. 1: Data structure and examples of OneSketch.

finding top-k frequent items and per-item frequency estimation in Section III-D and Section III-E, respectively. The symbols frequently used in this paper are shown in Table I.

A. The OneSketch Structure

As shown in Fig. 1, OneSketch consists of two part: a *Heavy Part* and a *Light Part*, which are designed to accurately record the frequencies of top-k items and infrequent items, respectively.

The Heavy Part is a hash table with n buckets $\mathcal{B}[1]$, $\mathcal{B}[2]$, $\cdots$, $\mathcal{B}[n]$, and associated with the hash function $h(.)$. Each bucket consists of d cells, each of which stores two fields: item ID (key) and its count C_H. For convenience, we use $\mathcal{B}[i][j]$ to represent the j^{th} cell in the i^{th} bucket.

The Light Part is a tailored Tower_CU [30] (see Section II-A). Each Tower_CU has two arrays: $\mathcal{A}[1]$ with $2 * w$ 2-bit counters and $\mathcal{A}[2]$ with w 4-bit counters, each of which is associated with a pair-wisely independent hash function $g_s(.)$ ($s = 1$ or 2). Each counter only records the count value of the item. Therefore, the 2-bit counter will overflow if the count exceeds **3**, and the 4-bit counter will overflow if the count exceeds **15**. For convenience, we use $\mathcal{A}[s][t]$ to represent the t^{th} counter in the s^{th} array.

B. Design Philosophy and Techniques

The design philosophy of OneSketch is **Overestimation Control**, *i.e.*, reducing the overestimated frequency as much as possible to approach the real frequency on the premise of ensuring overestimation. Towards the design goal, we propose four key techniques as follows.

1) **Fine-Grained Control (Light).** We find that the 8-bit counters used in ES can be further compressed to 2-bit or 4-bit, resulting in $2 \sim 4$ times more counters with the same space, which means $2 \sim 4$ times fewer hash collisions and errors. By combining the lower half of TowerSketch [30] with our proposed techniques, we achieve the aforementioned counter compression without significant additional overflow errors.
2) **Frequency Read/Write Control (Heavy&Light).** When each item replacement occurs in the Heavy Part, the new item that succeeds in the item replacement should immediately `Read` (merge) its frequency recorded in the Light Part and set it as the original value in the Heavy Part. This technique avoids possible overestimation errors in the Light Part due to hash collisions as soon as possible in the future, and essentially addresses well the sources of error in ES as described in Section I-B. Accordingly, the Light Part should also be `written` with the frequency of the least frequent item that is kicked from the Heavy Part.
3) **Repeat Control (Heavy&Light).** Followed by the above *Frequency Read/Write Control* technique, the item may go through the process of being written/kicked to the Light Part firstly, read back to the Heavy Part and then written/kicked to the Light Part again, which will cause the issue of repeated accumulations in the Light Part. Therefore, we improve the insertion algorithm of the Light Part, which only records the maximum value between written/kicked value and counter value to avoid repeated writing in the Light Part in order to further reduce overestimation errors when using the *Frequency Read/Write Control* technique.
4) **Replacement Control (Heavy).** We observe that, if the `Read` value of the Light Part does not overflow and is less than the least frequent item in the Heavy Part, replacement is meaningless and wasteful. Thereafter, unlike the one-step replacement decision in SOTA scheme, we propose the first double-check replacement strategy.

C. Design of Light Part

1) Insertion of Light Part:

Rationale: Thanks to our *Frequency Read/Write Control* technique (detailed later), an item ends up only in the Heavy Part or in the Light Part, not both. When *Frequency Read/Write Control* occurs, although the estimated value in the Light Part is merged to the Heavy Part, there is still a backup in the Light Part. It means that the estimated value in the Light Part will be repeatedly accumulated if item replacement and further *Frequency Read/Write Control* occur several times for an item. For example, assuming that the replacement path of item e is $Heavy\ Part \rightarrow Light\ Part \rightarrow Heavy\ Part \rightarrow Light\ Part$, when the second time $Heavy\ Part \rightarrow Light\ Part$ (item replacement) occurs, C_H of item e includes C_L queried in the last $Light\ Part \rightarrow Heavy\ Part$ (*Frequency Read/Write Control*) procedure. Thus the second occurrence of $Heavy\ Part \rightarrow Light\ Part$ will result in twice repeated accumulations for C_L. As a consequence, N occurrences of item replacements will result in N times repeated accumulations, leading to large overestimation in the Light Part.

To address this problem effectively, we propose a new technique called **Repeat Control**. The goal is to reduce the overestimation of OneSketch by optimizing the traditional insertion strategy of Tower_CU, which updates the count value of each counter to $min\{max\{\mathcal{A}[i][g_i(e)], C_L+C\}, 2^{\delta} -$

$1\}(i = 1, 2)$. Our optimized insertion strategy works as follows: when inserting an item e with the count value C, if C is equal to 1, we just increment the smallest counter(s) that are not overflowed by 1 using CU [32] insertion strategy. Otherwise, we update the count value of each counter to $min\{max\{\mathcal{A}[i][g_i(e)], max\{C_L, C\}\}, 2^\delta - 1\}(i = 1, 2)$. Using $max\{C_L, C\}$ instead of $C_L + C$ means that there is no need to update if the count value C is less than C_L, which will lead to a more tightly overestimated value.

It should be noted that, if a δ-bit counter overflows after the update, we will set its value to $2^\delta - 1$ and regard it as an overflowed counter. For an overflowed counter, we consider its count value as $+\infty$, which cannot be incremented. It means that the maximum value of a δ-bit counter is $2^\delta - 2$.

2) Query of Light Part:

The query procedure of the Light Part returns the minimum value of the hashed counters $\mathcal{A}[s][g_s(e)]$. Note that the value of an overflowed counter is $+\infty$. If all counters in the Light Part are overflowed, it will return **15** and the query value C_L will be regarded as an overflowed value.

Algorithm 1: Insertion of OneSketch

```
Input: Incoming item $e$
if $e$ is in one cell of $\mathcal{B}[h(e)]$ then
    $C_H + +$;
    return;
if $\mathcal{B}[h(e)]$ has an empty cell then
    Set the empty cell to $< e, 1 >$;
    return;
if we use the top-$k$ version then
    ReplaceforTopK(e);
else
    ReplaceforPerItem(e);
```

Algorithm 2: `ReplaceforTopK(e)`

```
if the probability $\mathcal{P} = \frac{1}{C'_H+1}$ does not hold then
    Insert $< e, 1 >$ into the Light Part;
else
    $ID \leftarrow e$;
    $C_H \leftarrow C_L + 1$;
    Insert $< e', C'_H >$ into the Light Part;
```

D. Operations (Top-K Version)

1) Insertion of OneSketch :

The pseudo-code of the insertion procedure is shown in Algorithm 1. Note that `ReplaceforTopK(e)` (Algorithm 2) is the top-k version of the item replacement procedure shown in this section, while `ReplaceforPerItem(e)` (Algorithm 3) is the per-item version in Section III-E.

Initially, all ID fields are set to null, and all count fields are set to 0. Given an incoming item with ID e, it is first mapped to the bucket $\mathcal{B}[h(e)]$ in the Heavy Part by computing the hash function $h(e)$ $(1 \leq h(e) \leq n)$. According to the information of $\mathcal{B}[h(e)]$, there are three cases as follows.

Case 1: e is in one cell of $\mathcal{B}[h(e)]$. OneSketch increments the count field C_H in the cell by 1.

Case 2: e is not in $\mathcal{B}[h(e)]$, but there is at least one empty cell in $\mathcal{B}[h(e)]$. OneSketch inserts e into an arbitrary empty cell, and sets the ID field to e and sets C_H to 1.

Case 3: e is not in $\mathcal{B}[h(e)]$, and there is no empty cell. OneSketch tries to replace the item e' with the minimum count value C'_H in $\mathcal{B}[h(e)]$ with probability $\mathcal{P} = \frac{1}{C'_H+1}$ (equation from [49]). There are two *sub-cases*:

① If the probability condition does not hold, e no longer replaces e', but is inserted into the Light Part.

② If the probability condition holds, e successfully replaces e'. OneSketch sets the ID field to e, and evicts e'. Then, we propose a novel technique for the challenge-successful item e called **Frequency Read/Write Control**, to avoid possible overestimation errors in the Light Part due to hash collisions in the future and accurately estimate the item e. OneSketch queries the count of e in the Light Part: it reports C_L among the hashed counters $\mathcal{A}[s][g_s(e)]$, then sets the C_H of e in the Heavy Part to $C_L + 1$. OneSketch also inserts the evicted item e' and its C'_H into the Light Part.

Example 1 (Fig. 1): **(1)** For incoming item e_1, OneSketch maps it to bucket $\mathcal{B}[h(e_1)]$. Since there is a cell storing e_1, OneSketch increments its count from 6 to 7. **(2)** For incoming item e_3, OneSketch maps it to bucket $\mathcal{B}[h(e_3)]$. Since e_3 does not exist in $\mathcal{B}[h(e_3)]$, but there is still an empty cell in $\mathcal{B}[h(e_3)]$, OneSketch sets the ID field of the empty cell to e_3, and sets the count field to 1. **(3)** For incoming item e_7, OneSketch maps it to bucket $\mathcal{B}[h(e_7)]$. e_7 does not exist in $\mathcal{B}[h(e_7)]$, and there is no empty cell. Therefore, OneSketch tries to replace the least frequent item e_6 with e_7 with probability $\mathcal{P} = \frac{1}{4+1} = 0.2$. We assume that the probability condition holds, so e_7 successfully replaces e_6. OneSketch sets the ID field to e_7 and the count field as follows: OneSketch maps e_7 to the counters $\mathcal{A}[1][g_1(e_7)]$ and $\mathcal{A}[2][g_2(e_7)]$, and reports the minimum value of 2 between them. Thus, the count field is set to $2 + 1 = 3$.

2) Query of OneSketch :

Since the item ends up only in the Heavy Part or Light Part, the error introduced by the back-and-forth passing of the counts in the two parts as in ES is completely avoided, and the procedure of insertion and query are simplified. To query an item e, OneSketch first checks the Heavy Part in bucket $\mathcal{B}[h(e)]$. If e matches a cell in $\mathcal{B}[h(e)]$, it reports the corresponding count C_H. If e matches no cell, it reports C_L among the hashed counters $\mathcal{A}[s][g_s(e)]$ in the light part.

E. Operations (Per-Item Version)

Note that the operations in this section are only applicable to *per-item frequency estimation*, and we only show the description of **Case 3** (sub-case ②), which is different from the operations of top-k version in Section III-D1. The pseudo-code of the item replacement procedure optimized in this section is shown in Algorithm 3.

② If the probability condition holds, OneSketch first applies the partial **Frequency Read/Write Control** technique to query the count of e in the Light Part: it reports C_L among the hashed counters $\mathcal{A}[s][g_s(e)]$. Then, we propose another novel technique, called **Replacement Control**, to avoid meaningless item replacement. Specifically, OneSketch checks the value of C_L: if C_L is not overflowed and less than C'_H, it is considered that e has not successfully replaced e', and the remaining operations are the same as **Case 3** (sub-case ①) in Section III-D1; otherwise, e successfully replaces e'. OneSketch sets the ID field to e, and evicts e': OneSketch sets the C_H of e in the Heavy Part to $\max\{C_L, C'_H\} + 1$. Further, OneSketch inserts the evicted item e' and its C'_H into the Light Part.

Algorithm 3: `ReplaceforPerItem(e)`

```
if the probability $\mathcal{P} = \frac{1}{C'_H+1}$ does not hold then
    Insert $< e, 1 >$ into the Light Part;
else
    if $C_L$ is not overflowed and less than $C'_H$ then
        Insert $< e, 1 >$ into the Light Part;
    else
        $ID \leftarrow e$;
        $C_H \leftarrow max\{C_L, C'_H\} + 1$;
        Insert $< e', C'_H >$ into the Light Part;
```

Example 2: This example is an extended version of Example 1-**(3)** based on the above operations, and we still assume that the probability condition holds. (1) We assume: $\mathcal{A}[1][g_1(e_7)] = 2$, $\mathcal{A}[2][g_2(e_7)] = 6$ and $C'_H = 4$. Since C_L, equal to 2, has not overflowed and is less than C'_H, e_7 will still not replace e_6 although the probability condition holds. Instead, we insert e_7 into the Light Part directly. (2) We assume: $\mathcal{A}[1][g_1(e_7)] = 3$, $\mathcal{A}[2][g_2(e_7)] = 15$ and $C'_H = 4$. Since $\mathcal{A}[s][g_s(e_7)]$ both overflow and C_L is equal to 15, e_7 successfully replaces e_6. Then, OneSketch sets the ID field to e_7 and the count field to $15 + 1 = 16$. Further, OneSketch inserts the evicted item e_6 and its C'_H into the Light Part.

IV. Mathematical Analysis

In this section, we propose the mathematical analysis of OneSketch. We limit our results to the top-k version. First, we present a theorem about the query results of the Light Part in Section IV-B. We derive the formula of the error bound in Section IV-C.

A. Preliminary

Let $S = \{e_1, e_2, \dots, e_T\}$ be a data stream that contains T items, where $e_t \in \{1, 2, \dots, m\}$ appears at time t. Let f_i be the frequency of item i in the entire data stream S and $f_{(i,t)} = \sum_{k=1}^{t} 1_{\{e_k=i\}}$ be the frequency of item i at time t, we have $f_i = f_{(i,T)}$. At time t, define $\tau(i,t) \in [0,t]$ as the last time item i was not stored in the Heavy Part, *i.e.*, $\tau(i,t)+1$ was the last time when item i successfully replaced another item and was inserted into the Heavy Part or $\tau(i,t) = t$ if item i was not stored in the Heavy Part. Here we let $f_{(i,0)} = 0$ and if the item was always stored in the Heavy Part, $\tau(i,t) = 0$. Then, we let $H_{(i,t)} = f_{(i,t)} - f_{(i,\tau(i,t))}$ be the number of items inserted into the Heavy Part with ID i between $\tau(i,t)$ and t (note that $H_{(i,t)} \neq 0$ if and only if item i is stored in the Heavy Part at time t), let $L_{(i,t)} = f_{(i,t)} - H_{(i,t)}$ be the number of items inserted into the Light Part with ID i at time t and let $\widehat{L}_{(i,t)}$ be the query value of the light part. In the top-k version of OneSketch, we have $C_H = H_{(i,t)} + \widehat{L}_{(i,\tau(i,t))}, L_{(i,t)} = L_{(i,\tau(i,t))}$. Therefore, let $H_i = H_{(i,T)}, t_i = \tau(i,T)$, the true frequency of item i can be written as

$$f_i = H_i + L_{(i,t_i)}$$

and the estimated frequency can be written as

$$\widehat{f}_i = H_i + \widehat{L}_{(i,t_i)}$$

Note that according to the above analysis, the estimation error of OneSketch only comes from the Light Part.

B. Properties of the Light Part

Based on Section III, the Light Part is Tower_CU. Suppose that the Light Part contains l arrays. The i^{th} array has w_i counters and each counter consists of δ_i bits. We have $0 = \delta_0 < \delta_1 < \dots < \delta_l, w_1 > w_2 > \dots > w_l$. Based on the above definition, we have the following theorem about the query result of the Light Part:

Theorem IV.1. *In the top-k version of OneSketch, for $\forall i, t$, if $L_{(i,t)} < 2^{\delta_l} - 1$, then no under-estimation error occurs at time t and $\forall k \in [1,l], L_{(i,t)} \leqslant \mathcal{A}[k][g_k(i)] \leqslant \sum_{j=1}^{m} I_{g_k(i)=g_k(j)} L_{(j,t)}, (\widehat{L}_{(i,t)} = \min_{k=1}^{l} \mathcal{A}[k][g_k(i)])$. Here, we denote $\sum_{j=1}^{m} I_{g_k(i)=g_k(j)} L_{(j,t)}$ as C.*

Proof. *For an arbitrary item i, assume that for $\forall t < T_i, L_{(i,t)} < 2^{\delta_l} - 1$. We prove that the theorem holds at any point in time $t < T_i$ through induction. Given a mapped counter $\mathcal{A}[k][g_k(i)]$, with some other items mapped to this counter. Initially, item i is not in the Light Part and all the corresponding expressions in the inequality are 0, so the theorem holds. At any point in time t, there are five cases as follows.*

Case 1: *An item was inserted into the Heavy Part. we can deduce that $L_{(i,t)}, \mathcal{A}[k][g_k(i)], C$ stays the same and the theorem holds.*

Case 2: *An item $j \neq i$ was inserted into the Light Part. If $g_k(i) \neq g_k(j)$, then as for **Case 1**, the theorem holds. Otherwise, $C = C + 1$ and $\mathcal{A}[k][g_k(i)]$ increases at most by 1, therefore the theorem holds.*

Case 3: *An item i was inserted into the Light Part. We have $L_{(i,t)} = L_{(i,t-1)} + 1, C = C + 1$, according to the properties of the CU insertion, $\mathcal{A}[k][g_k(i)]$ increases by 1 or $L_{(i,t-1)} \leqslant \min_{k=1}^{l} \mathcal{A}[k][g_k(i)] < \mathcal{A}[k][g_k(i)]$, so $L_{(i,t)} \leqslant \mathcal{A}[k][g_k(i)]$ and the theorem holds.*

Case 4: *An item $j \neq i$ was evicted from the Heavy Part after $e_t \neq j$ arrives. If $g_k(j) \neq g_k(i)$, the theorem holds, otherwise the insertion in the Light Part is equivalent to $\max(C'_H - C_L, 0)$, therefore $L_{(i,t)}$ stays the same, $\mathcal{A}[k][g_k(i)]$ becomes $\max(C'_H, \mathcal{A}[k][g_k(i)])$ and $C = C + H_{(j,t)}$. Since $C'_H = \widehat{L}_{(j,\tau(j,t))} + H_{(j,t)}$, $\mathcal{A}[k][g_k(i)] = \mathcal{A}[k][g_k(j)] \geqslant \widehat{L}_{(j,t-1)} \geqslant \widehat{L}_{(j,\tau(j,t))}$ we can deduce that $C'_H - \mathcal{A}[k][g_k(i)] = \widehat{L}_{(j,\tau(j,t))} - \mathcal{A}[k][g_k(i)] + H_{(j,t)} \leqslant H_{(j,t)}$, and $\mathcal{A}[k][g_k(i)]$ increases at most by $H_{(j,t)}$. Therefore the theorem holds.*

Case 5: *Item i was evicted from the Heavy Part after $e_t \neq j$ arrives. This case is similar to **Case 4** except*

$L_{(i,t)} = L_{(i,t-1)} + H_{(i,t)}$. *Since no item i was inserted into the Light Part between $\tau(i,t)$ and t, we have $C + H_{(i,t)} \geqslant \max(C'_H, \mathcal{A}[k][g_k(i)]) \geqslant C'_H = \widehat{L}_{(i,\tau(i,t-1))} + H_{(i,t)} \geqslant L_{(i,\tau(i,t-1))} + H_{(i,t)} = L_{(i,t-1)} + H_{(i,t)} = L_{(i,t)}$, therefore the theorem holds.*

In summary, the theorem holds at any point of time $t < T_i$. □

Note that based on the CU version of TowerSketch and our insertion strategy, the Light Part reports better result than the Tower_CU. Also, at any time t for any item i, if Tower_CU does not overflow and we only consider those items inserted into the Light Part, the query result is better than the CM version of TowerSketch with the same amount of counters in each layer.

C. Error Bound

Theorem IV.2. *In the top-k version of OneSketch, for $\forall i$, given an arbitrary positive number ϵ, suppose u satisfies $2^{\delta_{u-1}} - 1 \leqslant L_{(i,t_i)} < 2^{\delta_u} - 1 (1 \leqslant u \leqslant l$, and $u = l + 1$ if $L_{(i,t_i)} \geqslant 2^{\delta_l} - 1)$, we have*

$$\Pr\{\widehat{f_i} \leqslant f_i + \epsilon\} \geqslant 1 - \Pi_{k=u}^{l} \frac{\sum_{j=1}^{m} L_{(j,t_i)}}{\min(2^{\delta_k} - L_{(i,t_i)} - 1, \epsilon) w_k}$$

Proof. *According to Section IV-A, we have*

$$\Pr\{\widehat{f_i} \leqslant f_i + \epsilon\} = \Pr\{\widehat{L}_{(i,t_i)} \leqslant L_{(i,t_i)} + \epsilon\}$$

If $L_{(i,t_i)} \geqslant 2^{\delta_l} - 1$, then $\widehat{L}_{(i,t_i)} = 2^{\delta_l} - 1$ and the above probability becomes 1, therefore the theorem holds. Next we assume that the Light Part does not overflow and ϵ satisfies $L_{(i,t_i)} + \epsilon \leqslant 2^{\delta_l} - 1$

We define an indicator variable $I_{i,k,j}$ as

$$I_{i,k,j} = \begin{cases} 1, & g_k(i) = g_k(j) \wedge i \neq j \\ 0, & otherwise \end{cases}$$

As the l hash functions are independent from each other, we have:

$$E(I_{i,k,j}) = \Pr\{g_k(i) = g_k(j)\} = \frac{1}{w_k}$$

Based on the analysis in Section IV-B, at time t_i, we define another variable $X_{i,k} = \sum_{j=1}^{m} L_{(j,t_i)} \cdot I_{i,k,j}$ indicating an upper bound of the over-estimation error caused by hash collisions in counter $\mathcal{A}[k][g_k(i)]$. Here we only need to consider $k \geqslant u$ since $L_{(i,t_i)} \geqslant 2^{\delta_{u-1}} - 1$, and we have

$$\mathcal{A}[k][g_k(i)] \leqslant L_{(i,t_i)} + X_{i,k}$$

$$E(X_{i,k}) = E(\sum_{j=1}^{m} L_{(j,t_i)} \cdot I_{i,k,j}) = \sum_{j=1}^{m} L_{(j,t_i)} \cdot E(I_{i,k,j}) = \frac{\sum_{j=1}^{m} L_{(j,t_i)}}{w_k}$$

Therefore, according to the Markov inequality:

$$\begin{aligned}
&\Pr\{\widehat{L}_{(i,t_i)} \geqslant L_{(i,t_i)} + \epsilon\} \\
=&\Pr\{\forall k \geqslant u, \mathcal{A}[k][g_k(i)] \geqslant L_{(i,t_i)} + \epsilon\} \\
=&\Pi_{k=u}^{l} \Pr\{\mathcal{A}[k][g_k(i)] \geqslant \min(2^{\delta_k} - 1, L_{(i,t_i)} + \epsilon)\} \\
\leqslant&\Pi_{k=u}^{l} \Pr\{X_{i,k} \geqslant \min(2^{\delta_k} - L_{(i,t_i)} - 1, \epsilon)\} \\
\leqslant&\Pi_{k=u}^{l} \frac{E(X_{i,k})}{\min(2^{\delta_k} - L_{(i,t_i)} - 1, \epsilon)} \\
\leqslant&\Pi_{k=u}^{l} \frac{\sum_{j=1}^{m} L_{(j,t_i)}}{\min(2^{\delta_k} - L_{(i,t_i)} - 1, \epsilon) w_k}
\end{aligned}$$

Therefore, we have

$$\Pr\{\widehat{f_i} \leqslant f_i + \epsilon\} \geqslant 1 - \Pi_{k=u}^{l} \frac{\sum_{j=1}^{m} L_{(j,t_i)}}{\min(2^{\delta_k} - L_{(i,t_i)} - 1, \epsilon) w_k}$$

□

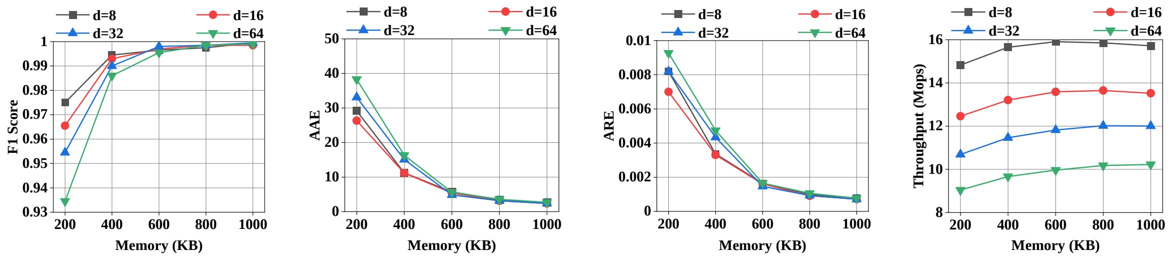
Note that according to Section III, in the Light Part of OneSketch, we have $l = 2, \delta_1 = 2, \delta_2 = 4, w_1 = 2 * w, w_2 = w$. Then, based on the above theorem we can conclude that with high probability, the over-estimation error of frequent items is less than ϵ (at most 15). Therefore, the relative error is very small, since the true frequency of a frequent item is usually large. For example, the frequency of the top 2000 items in the IP Trace Dataset we used for experiments in Section V is greater than 1000.

V. Experimental Results

A. Experimental Setup

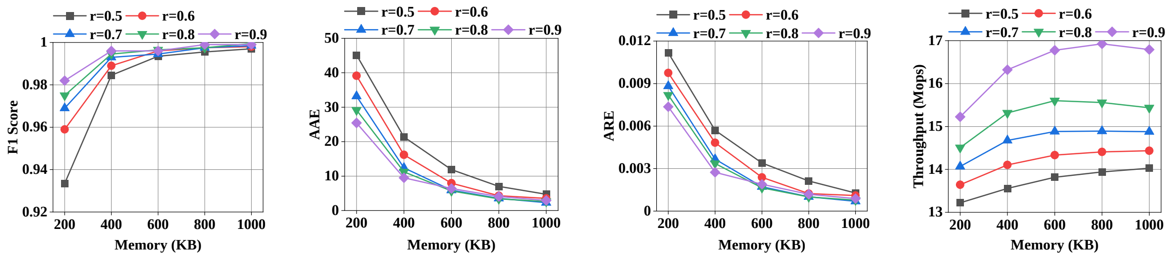
Implementation: We implement OneSketch and all other algorithms in C++. The hash functions are implemented using the 32-bit Bob Hash (obtained from the open-source website [50]) with different initial seeds. We list these 12 SOTA schemes and number them as follows: **[S1]** ES [14], [25]; **[S2]** MV [26], [43]; **[S3]** USS [38]; **[S4]** SS [37]; **[S5]** CM [29]; **[S6]** CU [32]; **[S7]** AS [39]; **[S8]** SALSA [20]; **[S9]** LLF [28]; **[S10]** FCMS [35]; **[S11]** C [33]; **[S12]** UM [42].

Algorithm Configuration: For OneSketch, the Heavy Part is array of buckets. Each bucket includes 8 cells and each cell has ID field and count field. The memory ratio of the Heavy Part and the Light Part is 4:1 for the top-k version and 1:4 for the per-item version. We use the per-item version for per-item frequency estimation and top-k version for the other four tasks. For ES, the heavy part is an array of buckets. Each bucket includes a vote- field and 8 cells. Each cell has ID field, count field and flag field. The light part is an array of 8-bit counters. The memory ratio of the heavy part and the light part is 1:3. For MV, we fix $r = 4$ and vary w according to the specified memory size. For USS and SS, the storage memory of each bucket is 100B and the number of buckets is determined by the memory size. For CM/CU/C, the number of array is 3. We use single CM/CU/C for per-item frequency estimation, and CM/CU/C with a minheap (CM/CU/C+heap) for the other four tasks which allocates 25% memory for sketch and 75% memory for minheap. The minheap is responsible for maintaining frequent items. For each item in the insertion process, if its frequency in CM/CU/C



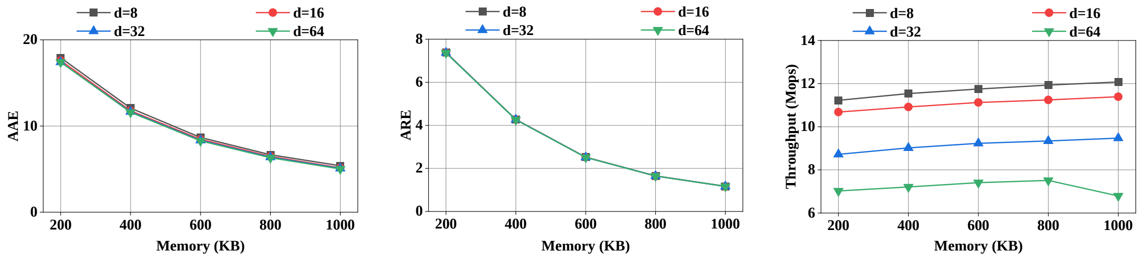
(a) F1 Score (b) AAE (c) ARE (d) Throughput

Fig. 2: Effects of the parameter d on frequency estimation for top-k frequent items, where $k = 2000$.



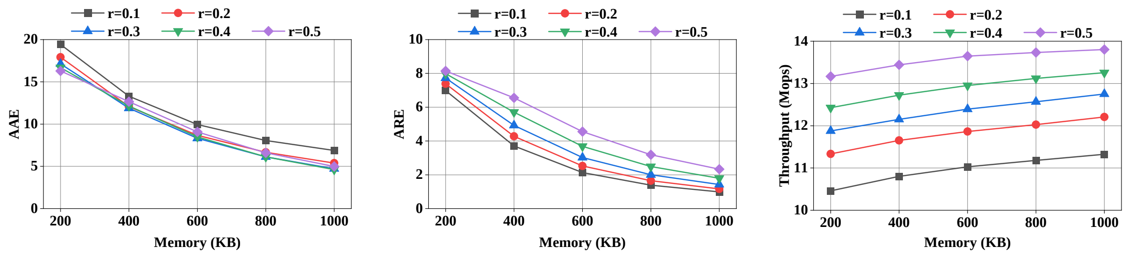
(a) F1 Score (b) AAE (c) ARE (d) Throughput

Fig. 3: Effects of the parameter r on frequency estimation for top-k frequent items, where $k = 2000$.



(a) AAE (b) ARE (c) Throughput

Fig. 4: Effects of the parameter d on frequency estimation for per-item.



(a) AAE (b) ARE (c) Throughput

Fig. 5: Effects of the parameter r on frequency estimation for per-item.

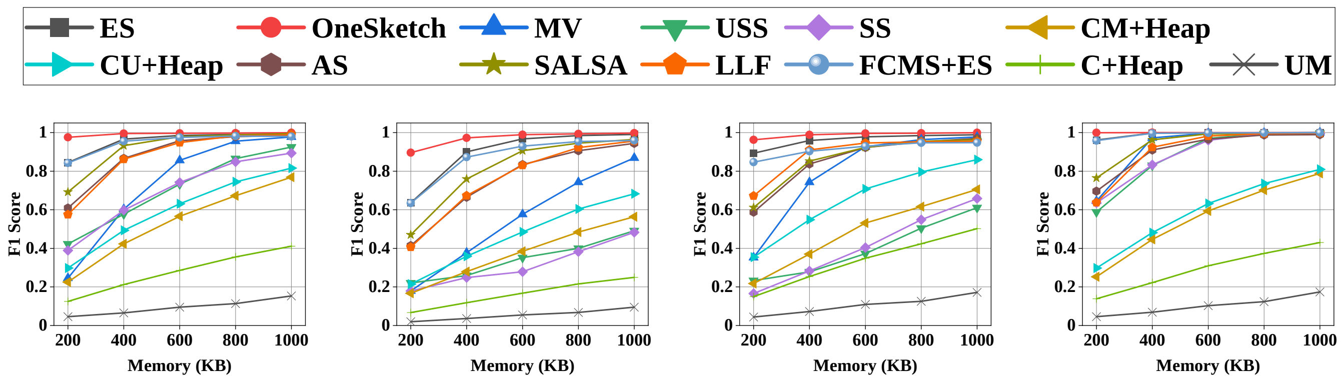
is larger than the minimum value of minheap, this item will be inserted into minheap. For AS, the filter includes 32 buckets and the rest of memory is for CM. Each bucket has ID field, new_count field and old_count field. For SALSA, we use the CM version. We set $d = 4$ and pick $s = 8$ bit counters as the default configuration. For LLF, we leverage CU and allocate 75% memory for it. We give 4 bits for each register, set the number of hash functions as 3 and threshold as 5. For FCMS, we use single FCM-Sketch (FCMS) for per-item frequency estimation, and FCM-Sketch with ES (FCMS+ES) for the other four tasks. We use configurations recommended by authors. For UM, we set the number of levels to 2.

Computation Platform: We conduct all the experiments on a 18-core CPU server (Intel i9-10980XE) with 128GB memory and 24.75MB L3 cache.

Datasets:

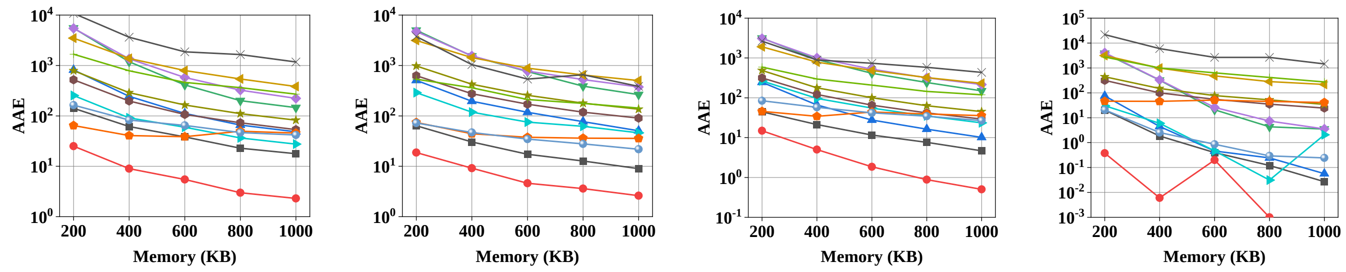
1) IP Trace Dataset. The IP Trace Dataset is streams of anonymized IP packets collected from high-speed monitors by CAIDA in 2018 [51]. We use the trace with a monitoring interval of 60s. Each item consists of a 5-tuple (13 bytes). There are around 27M items and 1.3M distinct items in this dataset.

2) MAWI Dataset. The MAWI Dataset is a set of anonymized traffic traces collected from trans-Pacific backbone link by



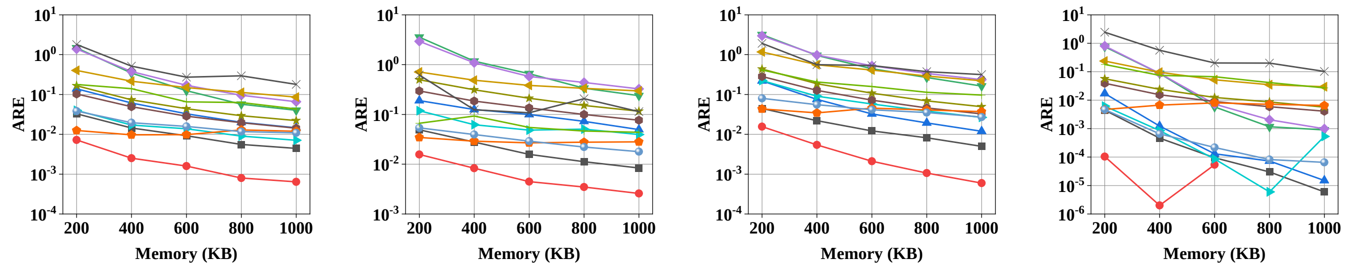
(a) IP Trace (b) MAWI (c) Network (d) Web Page

Fig. 6: F1 Score of frequency estimation for top-k frequent items.



(a) IP Trace (b) MAWI (c) Network (d) Web Page

Fig. 7: AAE of frequency estimation for top-k frequent items.



(a) IP Trace (b) MAWI (c) Network (d) Web Page

Fig. 8: ARE of frequency estimation for top-k frequent items.

MAWI Working Group [52]. Each item consists of a source IP (4 bytes) and a destination IP (4 bytes). There are around 17M items and 4.6M distinct items in this dataset.

3) Network Dataset. This dataset contains users' posting history on the stack exchange website [53]. Each item (4 bytes) represents the ID of each user. There are around 10M items and 0.7M distinct items in this dataset.

4) Web Page Dataset: The Web page Dataset is built from a collection of web HTML documents [54]. Each item (8 bytes) represents the number of distinct terms in a web page. There are around 32M items and 0.9M distinct items in this dataset.

Metrics:

1) Average Absolute Error: AAE $= \frac{1}{|\Psi|}\sum_{e_i \in \Psi} |f_i - \hat{f}_i|$, where f_i is the real frequency of item e_i, $\hat{f}_i$ is its estimated frequency, and Ψ is the total number of distinct items.

2) Average Relative Error: ARE $= \frac{1}{|\Psi|}\sum_{e_i \in \Psi} \frac{|f_i - \hat{f}_i|}{f_i}$, where f_i is the real frequency of item e_i, $\hat{f}_i$ is its estimated frequency, and Ψ is the total number of distinct items.

3) F1 Score: $\frac{2*RR*PR}{RR+PR}$, where Precision Rate (PR) refers to the ratio of true positive instances to all reported instances, and Recall Rate (RR) refers to the ratio of true positive instances to all actual instances.

4) Throughput: Million of operations (insertions) per second (Mops). We use throughput to measure the speed.

B. Experiments on Parameter Settings

In this section, we measure the effects of the key parameters of OneSketch based on the IP Trace Dataset, namely, the number of cells d per bucket in the Heavy Part, and the ratio r of the memory size of the Heavy Part to the memory size of the whole OneSketch. We use F1 Score (only for top-k items[2]), AAE, ARE, and Throughput to evaluate the effects.

1) Frequency Estimation for Top-K Items:

Effect of d (Fig. 2(a)-2(d)): *We find that the optimal value for d is 8.* In this experiment, we fix the ratio r to 0.8, and

[2]For the frequency estimation for per-item, it is equivalent to taking the k of the output top-k items as all items, so the F1 Score is all 1.

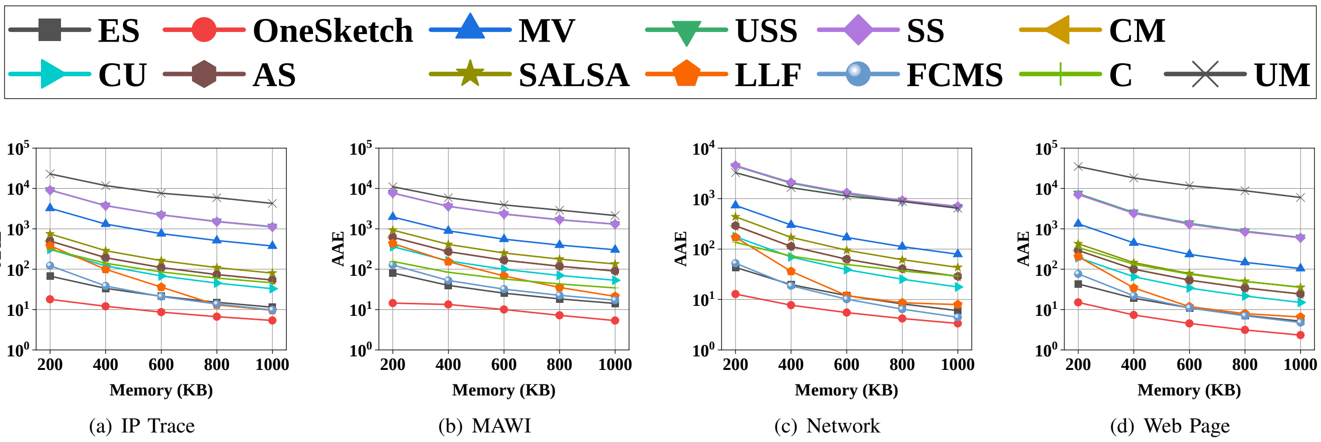


Fig. 9: AAE of frequency estimation for per-item.

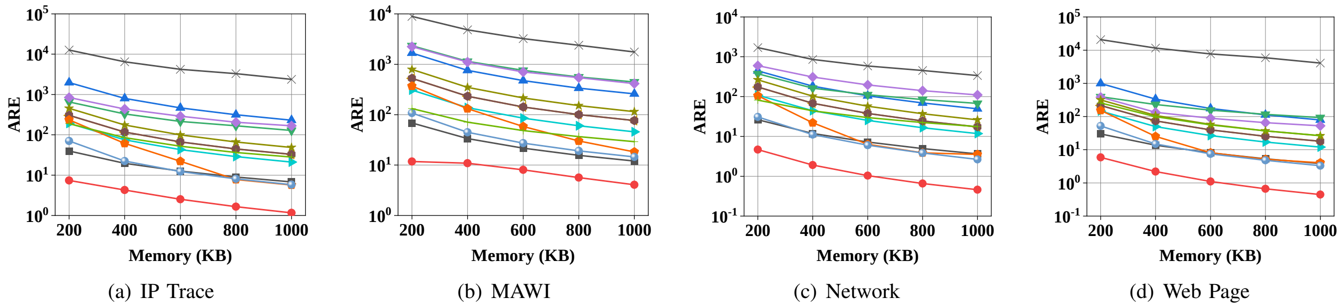


Fig. 10: ARE of frequency estimation for per-item.

vary d from 8 to 64. The results show that, especially when the memory size is relatively small, the F1 Score and throughput decrease as d increases, while AAE and ARE increase as d increases. Thus, we set $d = 8$.

Effect of the ratio r **(Fig. 3(a)-3(d)):** *We find that the optimal value for* r *is from 0.8 to 0.9.* In this experiment, we fix d to 8, and vary r from 0.5 to 0.9. The results show that the F1 Score and throughput increase as the ratio r increases, while AAE and ARE decrease as the ratio r increases. Therefore, the optimal value of the ratio r is from 0.8 to 0.9, and we set $r = 0.8$.

2) Frequency Estimation for Per-Item:

Effect of d **(Fig. 4(a)-4(c)):** *We find that the optimal value for* d *is 8.* In this experiment, we fix the ratio r to 0.2, and vary d from 8 to 64. The results show that the throughput decrease as d increases, while AAE and ARE do not change as d increases. For simplicity, we set $d = 8$.

Effect of the ratio r **(Fig. 5(a)-5(c)):** *We find that the optimal value for* r *is 0.2.* In this experiment, we fix d to 8, and vary r from 0.1 to 0.5. The results show that the ARE and throughput increase as the ratio r increases, and AAE does not change with increasing ratio r except when AAE is maximum at $r = 0.1$. To trade off ARE and throughput, we set $r = 0.2$. Since infrequent items in data streams predominate, the optimal value of the ratio r is different for top-k items and per-item frequency estimation.

C. Experiments on Five Measurement Tasks

In this section, we compare OneSketch with 12 SOTA schemes on five important measurement tasks: frequency estimation for top-k items (we default $k = 2000$) (Section V-C1), frequency estimation for per-item (Section V-C2), heavy hitters (Section V-C3), heavy changes in the time dimension (Section V-C4), heavy changes in the spatial dimension (Section V-C5), and throughput (Section V-C6). In summary, the results presented in Sections V-C1 to V-C5 are measured in terms of accuracy, while those in Section V-C6 are measured in terms of processing speed.

1) Frequency Estimation for Top-K Items:

F1 Score (Fig. 6(a)-6(d)): We find that, on the four datasets, the F1 Score of OneSketch is 12.5%, 73.0%, 59.2%, 61.6%, 74.3%, 66.7%, 38.2%, 32.4%, 38.6%, 13.8%, 83.8%, and 92.0% higher than that of **S1** to **S12** on average under 200KB of memory, respectively.

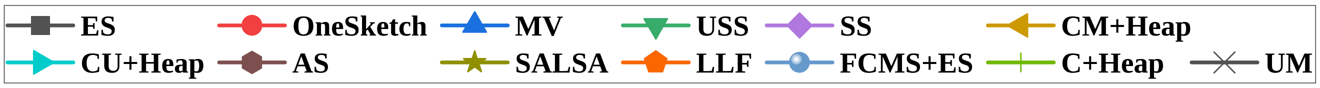
AAE (Fig. 7(a)-7(d)): We find that, on the four datasets, the AAE of OneSketch is 13.1, 19.3, 1946.7, 2047.7, 2304.5, 28.3, 247.9, 353.4, 103.0, 16.8, 2210.5, and 14997.2 times lower than that of **S1** to **S12** on average, respectively.

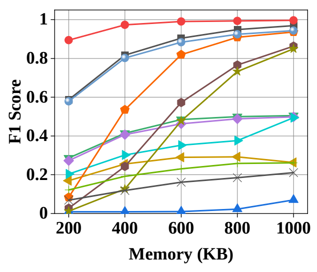
ARE (Fig. 8(a)-8(d)): We find that, on the four datasets, the ARE of OneSketch is 10.9, 14.4, 1463.6, 1546.0, 769.3, 20.5, 131.5, 193.0, 56.0, 14.4, 632.5, and 5637.1 times lower than that of **S1** to **S12** on average, respectively.

2) Frequency Estimation for Per-Item:

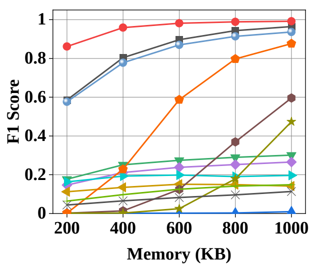
AAE (Fig. 9(a)-9(d)): We find that, on the four datasets, the AAE of OneSketch is 2.9, 78.9, 338.9, 336.0, 18.7, 11.5, 16.3, 28.1, 10.0, 3.4, 12.4, and 1061.1 times lower than that of **S1** to **S12** on average, respectively.

ARE (Fig. 10(a)-10(d)): we find that, on the four datasets, the ARE of OneSketch is 5.3, 142.8, 101.5, 116.0, 33.2, 21.3,

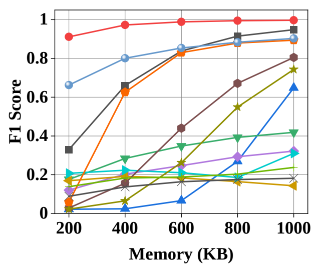




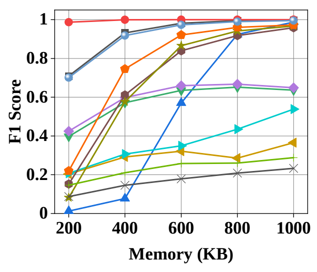
(a) IP Trace



(b) MAWI

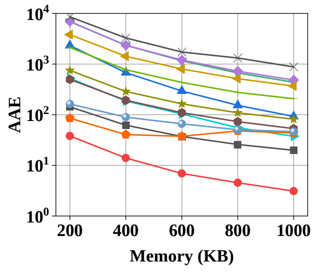


(c) Network

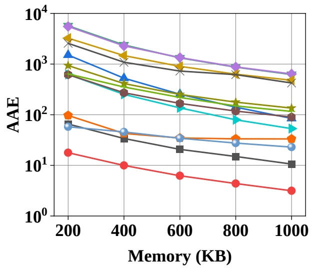


(d) Web Page

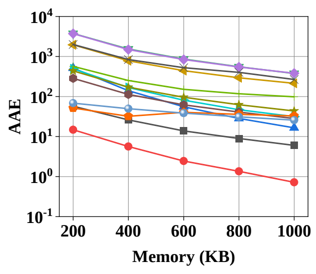
Fig. 11: F1 Score of heavy hitters.



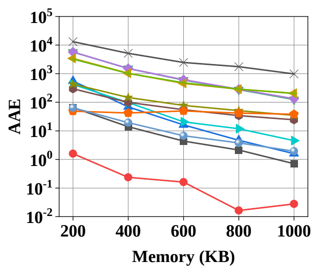
(a) IP Trace



(b) MAWI

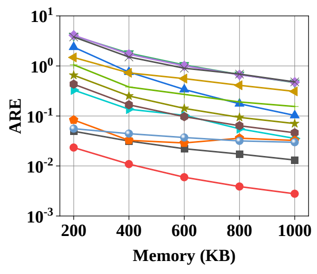


(c) Network

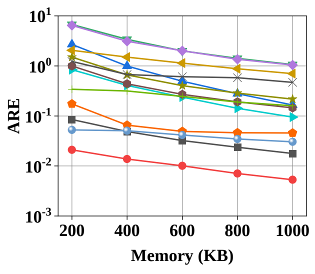


(d) Web Page

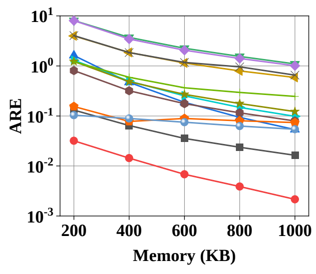
Fig. 12: AAE of heavy hitters.



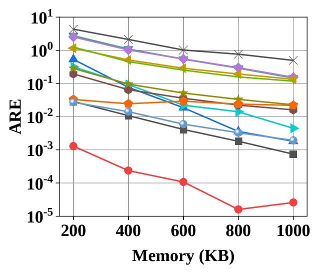
(a) IP Trace



(b) MAWI



(c) Network



(d) Web Page

Fig. 13: ARE of heavy hitters.

33.3, 49.7, 17.4, 6.6, 24.8, and 1883.3 times lower than that of **S1** to **S12** on average, respectively.

3) *Heavy Hitters:*

We set the threshold to be 2×10^{-5} the total size of traffic. As shown in Fig. 11-13, we find that the F1 Score, AAE, and ARE of OneSketch are always better than those of the 12 SOTA schemes.

F1 Score (Fig. 11(a)-11(d)): We find that, on the four datasets, the F1 Score of OneSketch is 36.0%, 90.2%, 65.4%, 67.2%, 75.0%, 71.8%, 86.1%, 88.4%, 82.3%, 28.4%, 79.6%, and 84.1% higher than that of **S1** to **S12** on average under 200KB of memory, respectively.

AAE (Fig. 12(a)-12(d)): We find that, on the four datasets, the AAE of OneSketch is 13.8, 114.8, 1194.0, 1202.2, 769.5, 86.6, 79.0, 116.5, 31.5, 16.7, 676.3, and 3024.3 times lower than that of **S1** to **S12** on average, respectively.

ARE (Fig. 13(a)-13(d)): We find that, on the four datasets, the ARE of OneSketch is 9.7, 144.2, 908.6, 863.3, 427.6, 89.1, 70.2, 103.6, 24.9, 11.6, 360.2, and 1386.2 times lower than that of **S1** to **S12** on average, respectively.

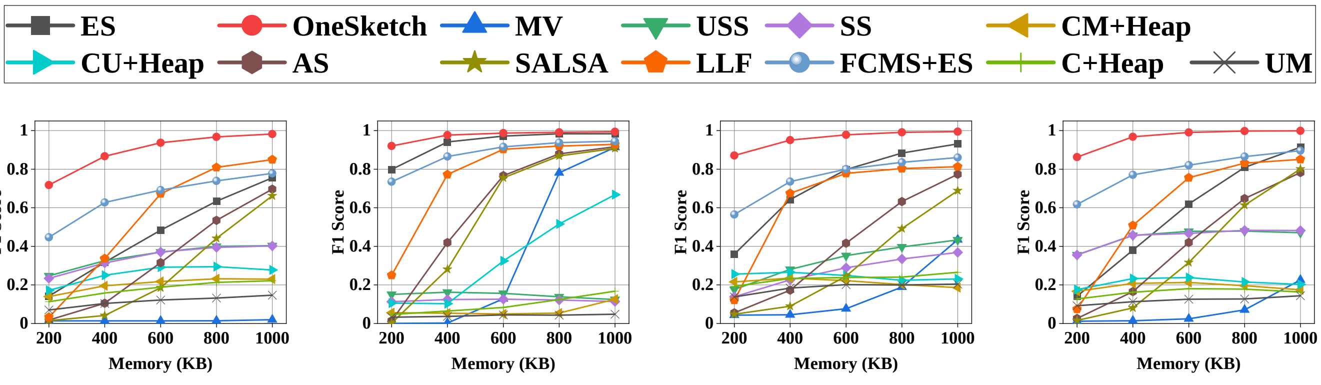
4) *Heavy Changes in the Time Dimension:*

We set the threshold to be 1×10^{-4} of the total size of traffic. The experimental results in Fig. 14 show that OneSketch always achieves a better F1 Score than the 12 SOTA schemes.

F1 Score (Fig. 14(a)-14(d)): We find that, on the four datasets, the F1 Score of OneSketch is 48.3%, 82.6%, 61.1%, 63.3%, 69.9%, 66.6%, 81.5%, 82.3%, 72.4%, 25.2%, 72.3%, and 76.0% higher than that of **S1** to **S12** on average under 200KB of memory, respectively.

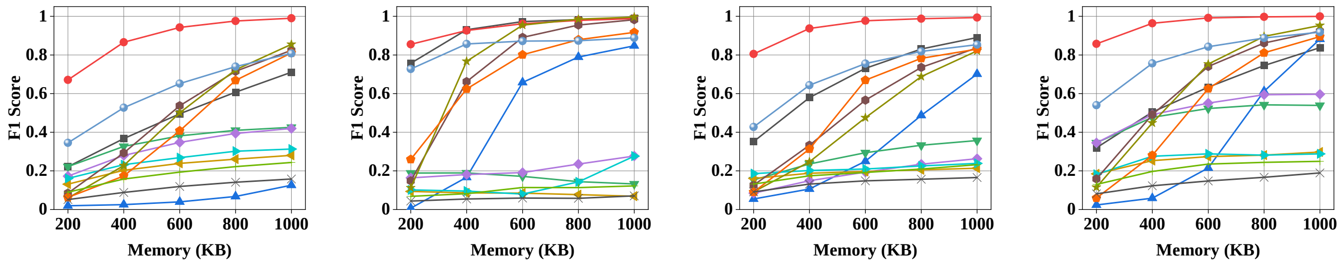
5) *Heavy Changes in the Spatial Dimension:*

In this section, we conduct experiments for the case where the frequency decreases sharply between two adjacent physical nodes as an example. The typical application is packet loss detection in networks. For the first node, we set each of our four datasets as the data streams which flow through it. For the second node, we reconstruct our datasets in the following steps: (1) We set the threshold to be 1×10^{-5} of the total size of traffic and pick up frequent items from the original dataset.



(a) IP Trace (b) MAWI (c) Network (d) Web Page

Fig. 14: F1 Score of heavy changes in the time dimension.



(a) IP Trace (b) MAWI (c) Network (d) Web Page

Fig. 15: F1 Score of heavy changes in the spatial dimension.

(2) For any item in the original frequent items, we drop it with 70% probability. Eventually, we set the threshold to be 1×10^{-4} of the total size of traffic to detect heavy changes in adjacent two nodes. The experimental results in Fig. 15 show that OneSketch always achieves a better F1 Score than the 12 SOTA schemes.

F1 Score (Fig. 15(a)-15(d)): We find that, on the four datasets, the F1 Score of OneSketch is 38.4%, 77.1%, 57.0%, 60.5%, 65.4%, 63.9%, 66.7%, 70.2%, 67.9%, 28.7%, 68.6%, and 73.0% higher than that of **S1** to **S12** on average under 200KB of memory, respectively.

6) Throughput:

In this section, we show the throughput of 13 schemes involving frequent-item frequency estimation tasks (Sections V-C1, and V-C3 to V-C5) and per-item frequency estimation tasks (Section V-C2). Because some of the schemes have two versions for these two types of tasks (see **Algorithm Configuration** in Section V-A for details), while other schemes have no distinction. The above results are shown as the average throughput of 13 schemes in memory 200KB to 1000KB stepping 200KB.

Throughput comparison of frequent-item versions (Fig. 16(a) - 16(d), orange part): We find that, on the four datasets, the throughput of OneSketch is 0.80, 1.6, 5.5, 1.6, 2.9, 3.6, 2.0, 1.3, 2.4, 0.79, 2.9, and 4.2 times higher than that of **S1** to **S12** on average, respectively. We can see that the throughput of OneSketch is less than ES because Frequency Read/Write Control technique will add extra operations when item replacement occurs and the insert operation of the tailored Tower_CU is much more complex than the light part of ES. What's more, for CM/CU/C, they use minheap to maintain top-k items so that their throughput decreases and is less than that of OneSketch.

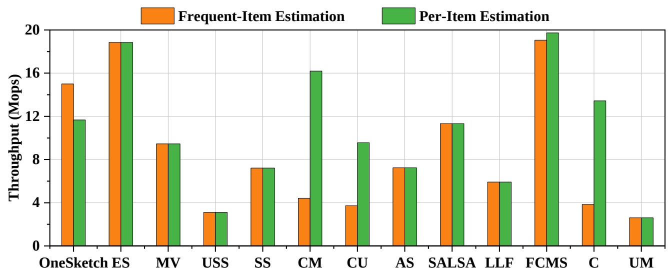
Throughput comparison of per-item versions (Fig. 16(a) - 16(d), green part): We find that, on the four datasets, the throughput of OneSketch is 0.61, 1.2, 4.2, 1.2, 0.72, 1.3, 1.6, 1.01, 1.9, 0.60, 0.69, and 3.2 times higher than that of **S1** to **S12** on average, respectively. For OneSketch, due to adding Replacement Control technique, the throughput decreases sightly. For CM/CU/C, compared with the top-k version, they implement without minheap so that throughput increases and may larger than OneSketch.

VI. Conclusion

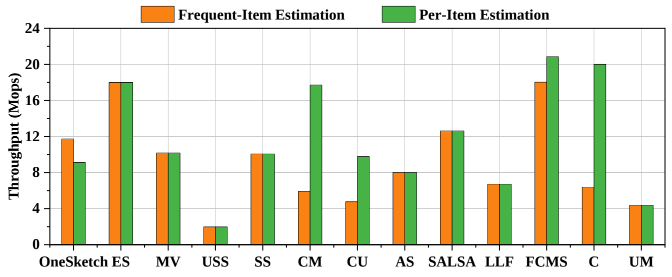
In this paper, we propose the OneSketch, which is generic for five important tasks and more accurate than SOTA solutions. One of the measurement tasks considered for OneSketch has hardly been studied despite having interesting applications. We call it heavy changes in the spatial dimension. The key design philosophy of OneSketch is overestimation control, around which we propose four techniques that embrace hash collisions and minimise overestimation errors in terms of extremely recurrent item replacements. Experimental results show that OneSketch performs better in five measurement tasks than ElasticSketch and the other 11 schemes.

Acknowledgment

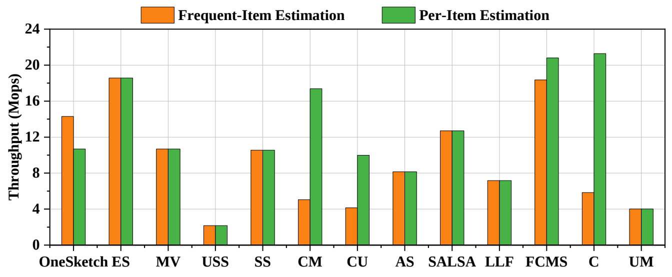
The authors would like to thank their editor(s), and the anonymous reviewers for their thoughtful feedback. The authors would also like to thank Xi Peng and Xin Xie from Theory Lab, Central Research Institute, 2012 Labs, Huawei



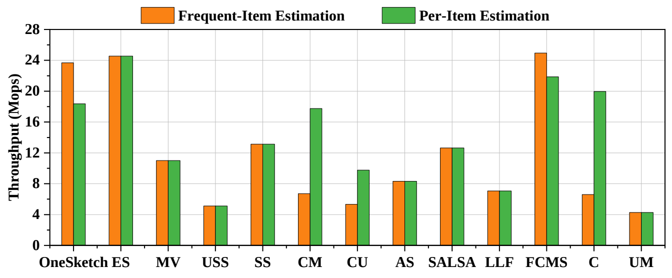
(a) IP Trace



(b) MAWI



(c) Network



(d) Web Page

Fig. 16: Throughput of tasks involving frequent-item and per-item frequency estimation for 13 schemes.

Technologies Co. Ltd., Hong Kong SAR, China, for useful discussions.

References

[1] A. Das, J. Gehrke, and M. Riedewald, “Approximate join processing over data streams,” in *Proc. SIGMOD*, 2003, pp. 40–51.

[2] M. Zhang, M. Hadjieleftheriou, B. C. Ooi, C. M. Procopiuc, and D. Srivastava, “On multi-column foreign key discovery,” *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 805–814, 2010.

[3] Y. Tong, L. Chen, Y. Cheng, and P. S. Yu, “Mining frequent itemsets over uncertain databases,” *Proc. VLDB Endow.*, vol. 5, no. 11, pp. 1650–1661, 2012.

[4] Q. Huang and P. P. Lee, “Toward high-performance distributed stream processing via approximate fault tolerance,” *Proc. VLDB Endow.*, vol. 10, no. 3, pp. 73–84, 2016.

[5] M. Garofalakis, J. Gehrke, and R. Rastogi, “Querying and mining data streams: you only get one look a tutorial,” in *Proc. SIGMOD*, 2002, pp. 635–635.

[6] C. Jin, K. Yi, L. Chen, J. X. Yu, and X. Lin, “Sliding-window top-k queries on uncertain streams,” *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 301–312, 2008.

[7] Z. Fan, Y. Zhang, T. Yang, M. Yan, G. Wen, Y. Wu, H. Li, and B. Cui, “Periodicsketch: Finding periodic items in data streams,” in *Proc. ICDE*, 2022, pp. 96–109.

[8] W. Wang, H. Yin, Z. Huang, Q. Wang, X. Du, and Q. V. H. Nguyen, “Streaming ranking based recommender systems,” in *Proc. SIGIR*, 2018, pp. 525–534.

[9] Y. Li, X. Wu, Y. Jin, J. Li, and G. Li, “Efficient algorithms for crowd-aided categorization,” *Proc. VLDB Endow.*, vol. 13, no. 8, pp. 1221–1233, 2020.

[10] T. Jin, K. Huang, J. Tang, and X. Xiao, “Optimal streaming algorithms for multi-armed bandits,” in *Proc. ICML*, 2021, pp. 5045–5054.

[11] Y. Wang, Y. Tong, D. Shi, and K. Xu, “An efficient approach for cross-silo federated learning to rank,” in *Proc. ICDE*, 2021, pp. 1128–1139.

[12] 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 *Proc. SIGCOMM*, 2017, pp. 113–126.

[13] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, “Evaluating the power of flexible packet processing for network resource allocation,” in *Proc. NSDI*, 2017, pp. 67–82.

[14] 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 *Proc. SIGCOMM*, 2018, pp. 561–575.

[15] R. Ramaswamy, L. Kencl, and G. Iannaccone, “Approximate fingerprinting to accelerate pattern matching,” in *Proc. IMC*, 2006, pp. 301–306.

[16] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. A. Dinda, M.-Y. Kao, and G. Memik, “Reversible sketches: enabling monitoring and analysis over high-speed data streams,” *IEEE/ACM Transactions on Networking*, vol. 15, no. 5, pp. 1059–1072, 2007.

[17] N. H. Park, S. H. Oh, and W. S. Lee, “Anomaly intrusion detection by clustering transactional audit streams in a host computer,” *Information Sciences*, vol. 180, no. 12, pp. 2375–2389, 2010.

[18] Y. Zhang, X. Lin, J. Xu, F. Korn, and W. Wang, “Space-efficient relative error order sketch over data streams,” in *Proc. ICDE*, 2006, pp. 51–51.

[19] A. Gkoulalas-Divanis, D. Vatsalan, D. Karapiperis, and M. Kantarcioglu, “Modern privacy-preserving record linkage techniques: An overview,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4966–4987, 2021.

[20] R. B. Basat, G. Einziger, M. Mitzenmacher, and S. Vargaftik, “Salsa: Self-adjusting lean streaming analytics,” in *Proc. ICDE*, 2021, pp. 864–875.

[21] B. Shi, Z. Zhao, Y. Peng, F. Li, and J. M. Phillips, “At-the-time and back-in-time persistent sketches,” in *Proc. SIGMOD*, 2021, pp. 1623–1636.

[22] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, “Pfabric: Minimal near-optimal datacenter transport,” in *Proc. SIGCOMM*, 2013, pp. 435–446.

[23] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao *et al.*, “Packet-level telemetry in large datacenter networks,” in *Proc. SIGCOMM*, 2015, pp. 479–491.

[24] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, “Deconstructing datacenter packet transport,” in *Proc. HotNets*, 2012, pp. 133–138.

[25] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, “Adaptive measurements using one elastic sketch,” *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2236–2251, 2019.

[26] L. Tang, Q. Huang, and P. P. C. Lee, “Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams,” in *Proc. INFOCOM*, 2019, pp. 2026–2034.

[27] Y. Zhou, T. Yang, J. Jiang, B. Cui, M. Yu, X. Li, and S. Uhlig, “Cold filter: A meta-framework for faster and more accurate stream processing,” in *Proc. SIGMOD*, 2018, pp. 741–756.

[28] P. Jia, P. Wang, J. Zhao, Y. Yuan, J. Tao, and X. Guan, “Loglog filter: Filtering cold items within a large range over high speed data streams,” in *Proc. ICDE*, 2021, pp. 804–815.

[29] 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.

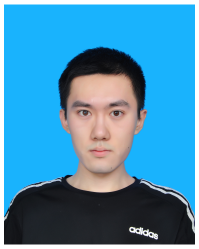
[30] K. Yang, Y. Li, Z. Liu, T. Yang, Y. Zhou, J. He, J. Xue, T. Zhao, Z. Jia, and Y. Yang, “Sketchint: Empowering int with towersketch for per-flow per-switch measurement,” in *Proc. ICNP*, 2021, pp. 1–12.

[31] "The source codes related to OneSketch." 2022. [Online]. Available: https://github.com/pkufzc/OneSketch
[32] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems*, vol. 21, no. 3, pp. 270–313, 2003.
[33] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," *Theor Comput Sci*, vol. 312, no. 1, pp. 3–15, 2002.
[34] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1622–1634, 2012.
[35] C. H. Song, P. G. Kannan, B. K. H. Low, and M. C. Chan, "Fcm-sketch: generic network measurements with data plane support," in *Proc. CoNEXT*, 2020, pp. 78–92.
[36] J. Misra and D. Gries, "Finding repeated elements," *Science of Computer Programming*, vol. 2, no. 2, pp. 143–152, 1982.
[37] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proc. ICDT*, 2005, pp. 398–412.
[38] D. Ting, "Data sketches for disaggregated subset sum and frequent item estimation," in *Proc. SIGMOD*, 2018, pp. 1129–1140.
[39] P. Roy, A. Khan, and G. Alonso, "Augmented sketch: Faster and more accurate stream processing," in *Proc. SIGMOD*, 2016, pp. 1449–1463.
[40] A. Desai, M. Ghashami, and J. M. Phillips, "Improved practical matrix sketching with guarantees," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, pp. 1678–1690, 2016.
[41] D. Thomas, R. Bordawekar, C. C. Aggarwal, and P. S. Yu, "On efficient query processing of stream counts on the cell processor," in *Proc. ICDE*, 2009, pp. 748–759.
[42] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proc. SIGCOMM*, 2016, pp. 101–114.
[43] L. Tang, Q. Huang, and P. P. C. Lee, "A fast and compact invertible sketch for network-wide heavy flow detection," *IEEE/ACM Transactions on Networking*, vol. 28, no. 5, pp. 2350–2363, 2020.
[44] V. Braverman and R. Ostrovsky, "Generalizing the layering method of indyk and woodruff: Recursive sketches for frequency-based vectors on streams," in *Proc. APPROX & RANDOM*, 2013, pp. 58–70.
[45] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *Proc. ESA*, 2003, pp. 605–617.
[46] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm," in *Proc. AofA*, 2007, pp. 137–156.
[47] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. EDBT*, 2013, pp. 683–692.
[48] R. S. Boyer and J. S. Moore, "Mjrty—a fast majority vote algorithm," *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pp. 105–117, 1991.
[49] R. Ben Basat, X. Chen, G. Einziger, R. Friedman, and Y. Kassner, "Randomized admission policy for efficient top-k, frequency, and volume estimation," *IEEE/ACM Transactions on Networking*, vol. 27, no. 4, pp. 1432–1445, 2019.
[50] "The source code of Bob Hash." 1997. [Online]. Available: http://burtleburtle.net/bob/hash/evahash.html
[51] "The CAIDA Anonymized Internet Traces." 2018. [Online]. Available: http://www.caida.org/data/overview/
[52] "MAWI Working Group Traffic Archive." 2010. [Online]. Available: http://mawi.wide.ad.jp/mawi/
[53] "The Network dataset Internet Traces," 2014. [Online]. Available: http://snap.stanford.edu/data/
[54] "Real-life Transactional Dataset." 2004. [Online]. Available: http://fimi.uantwerpen.be/data/

Zhuochen Fan received the Ph.D. degree in computer science from Peking University in 2023, advised by Prof. Tong Yang. He is currently working as a Boya Post-Doctoral Fellow with the School of Computer Science, Peking University. His research interests include sketches, network measurements, databases, and machine learning. He published papers in IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, ICDE, RTSS, ICPP, ICNP, *etc.*

Ruixin Wang is a M.E. candidate in Software Engineering from Peking University. His research interests include network measurements, sketches and machine learning.

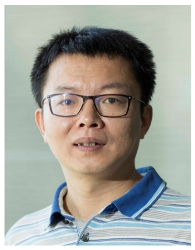
Yalun Cai is an undergraduate student of Peking University majoring in Computer Science. His research interests include network measurements and sketches.



Ruwen Zhang received the B.S. degree in mathematics from Peking University in 2021. He is currently pursuing the master's degree with Peking University, advised by Tong Yang. He has participated several articles in network area. He is interested in network and data stream processing.

Tong Yang (Member, IEEE) received the PhD degree in computer science from Tsinghua University in 2013. He visited the Institute of Computing Technology, Chinese Academy of Sciences (CAS). Now he is an associate professor with School of Computer Science, Peking University. His research interests include network measurements, sketches, IP lookups, Bloom filters, and KV stores. He has served as a TPC Member for several premier conferences such as INFOCOM and ICNP. He is currently an Associate Editor for *Knowledge and Information Systems*. He published dozens of papers in IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, *VLDB Journal*, SIGCOMM, SIGKDD, SIGMOD, NSDI, USENIX ATC, ICDE, VLDB, INFOCOM, *etc.*

Yuhan Wu received his bachelor degree in the Department of Electrical Engineering and Computer Science at Peking University in 2021. Currently he is a CS Ph.D. student in School of Computer Science at Peking University, advised by Tong Yang. His research interests lie in the fields of computer network and database, including key-value stores, network measurement, and sketches.



Bin Cui (Senior Member, IEEE) is a professor and Vice Dean in School of Computer Science at Peking University. He obtained his Ph.D. from National University of Singapore in 2004. His research interests include database system architectures, query and index techniques, big data management and mining. He is serving as Vice Chair of Technical Committee on Database China Computer Federation (CCF) and Trustee Board Member of VLDB Endowment. He was awarded Microsoft Young Professorship award (MSRA 2008), CCF Young Scientist award (2009), and Second Prize of Natural Science Award of MOE China (2014), *etc*.

Steve Uhlig obtained a Ph.D. degree in Applied Sciences from the University of Louvain, Belgium, in 2004. From 2004 to 2006, he was a Post-Doctoral Fellow of the Belgian National Fund for Scientific Research (F.N.R.S.). From 2004 to 2006, he was a visiting scientist at Intel Research Cambridge, UK, and at the Applied Mathematics Department of University of Adelaide, Australia. From 2006 to 2008, he was with Delft University of Technology, the Netherlands. Prior to joining Queen Mary, he was a Senior Research Scientist with Technische Universität Berlin/Deutsche Telekom Laboratories, Berlin, Germany. Since January 2012, he was the Professor of Networks and Head of the Networks Research group at Queen Mary University of London. From 2012 to 2016, he was a guest professor at the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. He's currently the Head of School of Electronic Engineering and Computer Science, QMUL.